平成 30 年度 公立はこだて未来大学卒業論文

DDoS 攻撃を行うマルウェアの IoT デバイス本体における検知手法の提案

水上 敬介

情報アーキテクチャ学科　1015237

指導教員　(主) 稲村 浩　(副) 中村 義隆

提出日 2019 年 1 月 29 日

Proposal of Detection Method in IoT Device of Executing DDoS Attack

by

Keisuke Mizukami

BA Thesis at Future University Hakodate, 2019

Advisor: Hiroshi. Inamura, Coadvisor: Yoshitaka. Nakamura

Department of Information Architecture
Future University Hakodate

January 29, 2019

Abstract– In recent years, IoT devices equipped with communication functions in various things are spreading explosively. As a result It is a social problem that a botnet is constructed by an IoT device infected with malware and a DDoS attack is performed. Malware called Mirai is published on the web site, and many variants of Mirai are made. In this research, we aim to detect malware that performs DDoS attack on IoT device , and to detect unknown malware. we pay attention to the fact that the variants of malware were make from published malware on web site, we detect malware that performs DDoS attack by determining whether there is a specific function of the original malware

Keywords: DDoS attack, IoT Device, malware, Mirai, Linux

概 要: 近年，世の中にある様々なものに通信機能を搭載した IoT 機器が爆発的に普及している．その結果，マルウェアに感染した IoT 機器によってボットネットが構築され大規模な DDoS 攻撃が行われ大きな問題となっている．その中でも，Mirai と呼ばれるマルウェアが Web 上で公開され，Mirai の亜種が多く作られている．本研究では，IoT デバイス本体において DDoS 攻撃を行うマルウェアを検知する手法を検討することによって，未知のマルウェアでも検知を行えることを目的とする．公開されているマルウェアを元に亜種が作成されていることに着目をして，オリジナルのマルウェアが持つ特定の関数の挙動を検出することによって DDoS 攻撃を行うマルウェアの検知を行う．

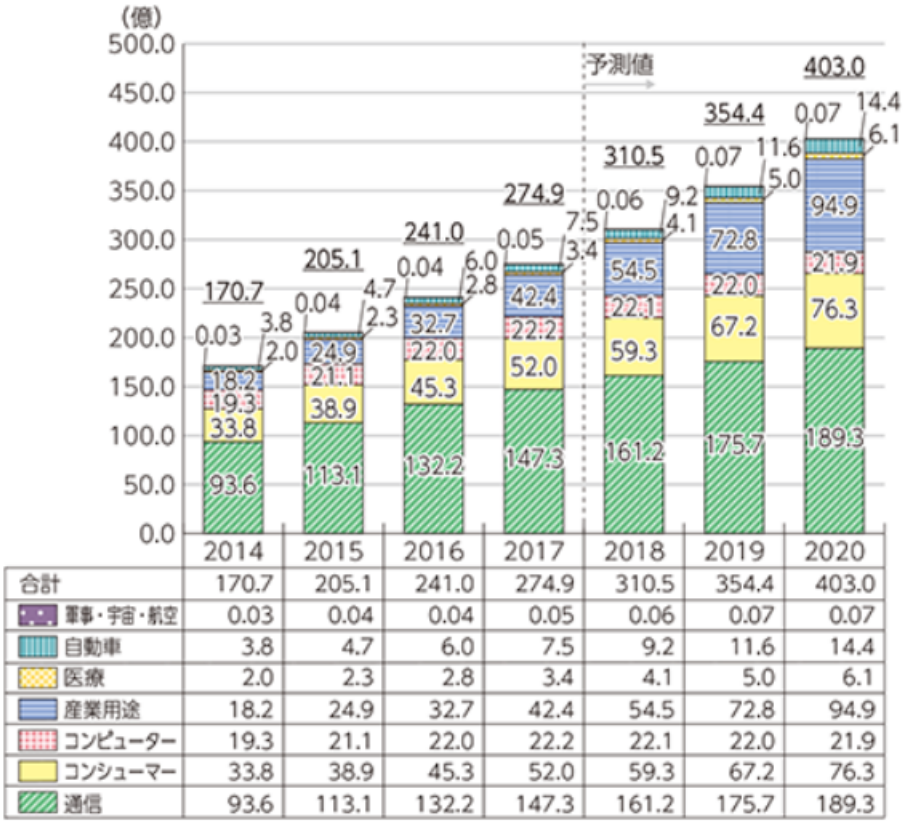
キーワード: DDoS 攻撃，IoT デバイス，マルウェア，Mirai

目 次

第1章　序論

1.1　背景

近年，インターネット技術やセンサー技術の進化を背景に，パソコンやスマートフォンなどのインターネット端末に加え，家電や自動車などの様々なものに通信機能を搭載したIoT デバイスが普及し始めている．総務省によると政界中の IoT デバイスの数は図 1.1 のように 2017 年時点で IoT デバイスが約 275 億台存在し，2020 年には IoT デバイスが 403 億台に及ぶと予想されている [1].



	2014	2015	2016	2017	2018	2019	2020
合計	170.7	205.1	241.0	274.9	310.5	354.4	403.0
軍事・宇宙・航空	0.03	0.04	0.04	0.05	0.06	0.07	0.07
自動車	3.8	4.7	6.0	7.5	9.2	11.6	14.4
医療	2.0	2.3	2.8	3.4	4.1	5.0	6.1
産業用途	18.2	24.9	32.7	42.4	54.5	72.8	94.9
コンピューター	19.3	21.1	22.0	22.2	22.1	22.0	21.9
コンシューマー	33.8	38.9	45.3	52.0	59.3	67.2	76.3
通信	93.6	113.1	132.2	147.3	161.2	175.7	189.3

図 1.1: 世界の IoT デバイス数の推移及び予測 (平成 30 年版情報通信白書 [1] より引用)

IoT デバイスを対象としたマルウェアが急増している．IoT デバイスの普及に伴う重要な問題の 1 つとしてセキュリティ問題が挙げられる．IoT デバイスのユーザ名やパスワードを初期設定の状態で使用する場合が多いことやデバイスの資源が限られていることから，セキュリティが十分に考慮されていない事がある．そのため，IoT デバイスを対象としたマルウェアが脅威となっている．その中でもネットワークサービスを停止させる深刻な問題を引き起こしているマルウェアには DDoS(Distributed Denial of Service) 攻撃を行っているものが多く存在し，その対策が重要視されている．DDoS 攻撃は，複数の他人のコンピュータを攻撃者が悪用し，公開されているサービスに大量のデータを送りつける事によって処理負荷を与えサービスを機能停止に追い込む攻撃である．代表的な DDoS 攻撃を行うマルウェアとして Mirai[2] が挙げられる．Mirai を利用して 2016 年 10 月に発生した，DNS サーバープロバイダである Dyn 社への DDoS 攻撃では IoT デバイスによるボットネットが利用され史上最大規模である 620Gbps の攻撃が観測された [3]．その後，Mirai のソースコード [4] が公開され，Wicked[5]，Satori[6]，Okiru[7] といった Mirai の亜種の開発が盛んに行われるようになった．2017 年には，Windows PC を踏み台にして感染可能な IoT デバイスを探索する Mirai によるネットワーク活動が観測された [8]．この Mirai は多くの端末に感染させることを目的としており，IoT デバイスを探索する機能に特化させた Mirai となっている．初期の Mirai よりも多くのポートをスキャンしログイン可能な端末を探索することにより Mirai を様々な IoT デバイスに拡散させることが可能になっている．Mirai や Mirai 亜種のマルウェアによって，多くの IoT デバイスが DDoS 攻撃に不正利用されるようになったことから，国立研究開発法人情報通信研究機構がパスワード設定などに不備のある IoT 機器の実態把握を目的として日本国内の IPv4 アドレスを対象に SSH，Telnet，HTTP である TCP の 22 番，23 番，80 番ポートを対象にポートスキャンを行った [9]．IoT デバイスの不正利用による DDoS 攻撃が問題視されており，社会的に注目されている．

1.2 IoT デバイス本体で検知を行う必要性

IDS(Instrusion Detection System) と呼ばれるマルウェアによる不正な通信やホストへの侵入，ファイルの改ざん等の不正な挙動の兆候を検出するシステムの設置場所としてネットワーク上に設置するネットワーク型と端末上に設置するホスト型の 2 種類が考えられる．ネットワーク型の IDS では，ネットワーク上に流れるデータを取得して解析し，不正と疑われるデータを検知したときには管理者に知らせる．ネットワークトラフィックから DDoS 攻撃を判別するのは難しく，誤検知する可能性が考えられる．しかし，ホスト型 IDS にて見られる，マルウェアに基づいて作成されたデータを用いたパターンマッチングによる検知手法では，誤検知率が低く既存のマルウェアを確実に検知できる利点が有る．公開されているソースコードを基に作成されたマルウェアは，オリジナルのマルウェアと共通するシグネチャが存在すると考えられるためパターンマッチングによる検知で亜種のマルウェアにも対応できると想定される．

脅威となっているマルウェアは，十分に管理が行われていない IoT デバイスで散見される，放置された初期パスワードのままのアカウントや，保守されていないシステムの脆弱性をついた攻撃を行うため，侵入されてしまうことは前提とすべきである．デバイス上で

検知をすることによって侵入したマルウェアによる不正な挙動の痕跡からパターンマッチングによる検知を行うことができる．そのため，デバイスの性能が限られている IoT デバイス上でマルウェアの検知を行う必要がある．

1.3 マルウェア解析

マルウェア解析とは，端末上にダウンロードされたマルウェアから内在している情報を得るための行為である．例えば，マルウェアが具備している機能を明らかにしたり，どういった組織を狙っているのかマルウェアが作成された目的を明らかにしたりするために行われる．マルウェア解析には，表層解析，動的解析，静的解析の 3 つのプロセスが存在している [10]．表層解析は，ファイル自体が悪性であるかどうかの情報収集や，ファイルのメタ情報の収集を行う．ファイルタイプや動作する CPU アーキテクチャの情報，ハッシュ値などをもとに，既知のマルウェアかどうか判定を行う．表層解析では，解析対象に対して解析ツールを用いて情報を収集する．マルウェアを動作させたり，マルウェア内のプログラムコードを分析したりはしない．動的解析は，マルウェアが実際に動作した際に端末上にどのような痕跡が残るのか，またどのような通信が発生するのかの情報収集を行う．ファイルやレジストリアクセスの情報を収集したり，通信を監視して通信先の IP アドレスやドメイン，URL，通信ペイロードといった情報を収集する．マルウェアを動作させるため，動的解析を行う環境は図 1.2 のように通常利用しているホストやネットワークから隔離しておくなど，安全性に対する十分な配慮が必要である．静的解析は，逆アセンブラやデバッガを用いてマルウェアのプログラムコードを分析し，具備されている機能や特徴的なバイト列など詳細な情報を収集する．動的解析で実行されなかったコードを分析して潜在的に保有している機能を明らかにしたり，マルウェア独自の通信プロトコルや通信先生成アルゴリズムのような動的解析だけでは特定が難しい情報の収集を行う．IoT デバイス上でマルウェアと判別するためには，疑わしいファイルを解析する必要がある．しかし，資源の限られた IoT デバイス上で静的解析を行い，疑わしいファイルをデバッガなどを用いてプログラムコードからマルウェアと判別する手法では，デバッガなどが IoT デバイス本来の動作を阻害してしまう可能性が高い．そのため，疑わしいファイルを解析する方法として動的解析や表層解析が候補として挙げられる．

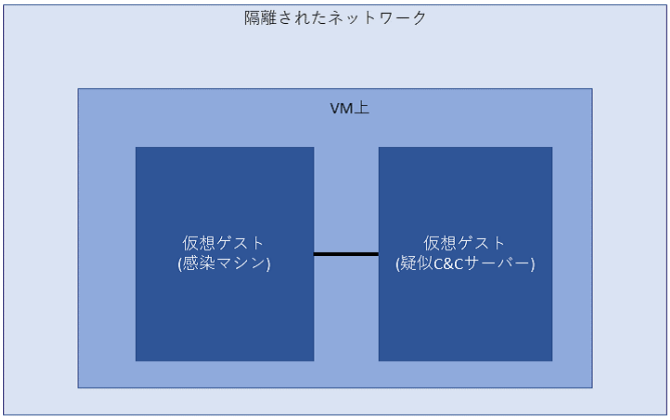


図 1.2: 動的解析環境

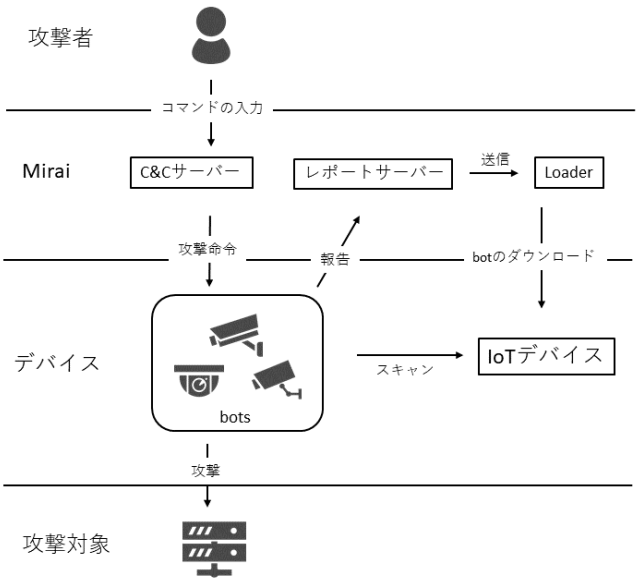


図 1.3: Mirai の概要図

1.4 マルウェア Mirai の概要

Mirai は，ネットワーク上で公開されている Linux で動作するデバイスを不正利用し，DDoS 攻撃を行うマルウェアである．ネットワークカメラやルータといった IoT デバイスをターゲットにしている．Mirai は，C&C サーバー，攻撃対象の管理データベース，Loader，bot の 4 つから構成される．Mirai の概要を図 1.3 に示す．

C&C サーバーは，ボットやユーザーから接続されるのを待機しており，主な機能として

ボット管理機能，ユーザー管理機能，攻撃指示機能がある．攻撃対象の管理データベースとして MySQL が利用されており，ユーザーのリストと攻撃履歴が記録されるようになっている．bot は，C&C サーバーからの攻撃命令を待機し，感染先でボットネットに加えられる新しいデバイスを探索するスキャン活動を行う．スキャン活動を行いログインできる端末を見つけた場合には，IP アドレス，ポート，ログイン情報をスキャンサーバーへと送る．感染経路について，Mirai は、Telnet ログインが可能な場合に感染する．Loader は，スキャン活動からレポートサーバーに送られた端末のログイン情報をもとに Telnet ログインを試みる．Telnet ログインに成功した際には，攻撃者が用意した http サーバーまたは tftp サーバーから，Mirai のバイナリファイルを対象の IoT デバイスにダウンロードし bot を実行させる．bot の動作後には，C&C サーバーとの通信を始め，C&C サーバーから送られてくる攻撃命令を受け取り，IoT デバイスが特定のサーバーに攻撃を始める．C&C サーバーによる DDoS 攻撃命令は表 1.1 のように 10 種類存在しており，攻撃を行った bot は再び C&C サーバーから攻撃命令が送られてくるのを待機している．

表 1.1: 攻撃種類

攻撃種類	詳細
UDP 攻撃	UDP パケットを大量に送信する
プレーン UDP 攻撃	高速化のために最適化を行い UDP パケットを大量に送信する
SYN 攻撃	SYN パケットを大量に送信する
ACK 攻撃	ACK パケットを大量に送信する
HTTP 攻撃	HTTP リクエストを大量に送信する
DNS リゾルバ攻撃	DNS サーバーへ大量の名前解決のためのリクエストを送信する
GRE IP 攻撃	GRE プロトコルによるパケットを大量に送信する
GRE イーサネット攻撃	イーサネットと GRE プロトコルによるパケットを大量に送信する
VSE 攻撃	ゲームエンジンに対して UDP パケットを大量に送信する
STOMP 攻撃	TCP セッション確立後に ACK パケットを大量に送信する

1.5 研究目的

本研究の目的は，IoT デバイス上で DDoS 攻撃を行うマルウェア Mirai とその亜種の未知のマルウェアの検知を行うことである．マルウェアの内部関数に着目をしたマルウェア検知手法を提案する．前項で述べたマルウェア解析方法である，表層解析，動的解析を行った 2 つの提案手法について述べる．表層解析を行ったマルウェア検知手法では，疑わしきバイナリファイルからシンボルテーブルと呼ばれるプログラム内で使用されている関数名，変数名を取得し，マルウェアが具備している関数の有無によってマルウェアの検出を行う．動的解析を行ったマルウェア検知手法では，マルウェアによって呼び出される内部関数によって呼び出されるシステムコール系列を用いてマルウェアの検出を行う。シンボルテーブルを用いた検知手法は，バイナリファイルのシンボルテーブルを削除するコマンドなどの検知対策が行われた場合には，マルウェアの検出をすることができない．しかし，システムコール系列を用いた検知を行うことによって，上記のような検知対策がなされた場合

でもマルウェアの検出が可能になる．

1.6　論文の構成

本論文では，6 章から構成されている．1 章では，本研究の背景や研究目的について述べる．2 章では，関連研究とその課題について述べる．3 章では，シンボルテーブルを用いた検知手法の提案についてシンボルテーブルを用いたマルウェアの検出方法について述べる．4 章では，システムコール呼び出し履歴を用いた検知手法について述べ，マルウェアによって呼び出されるシステムコール系列を用いたマルウェアの検出方法の課題点と解決方法について述べる．5 章では，提案システムによるマルウェアの検知精度について述べ，6 章でまとめる．

第2章　関連研究・技術

2.1　関連研究

2.1.1　DDoS 攻撃を行うマルウェアの分析

DDoS 攻撃を行う IoT デバイスを対象としたマルウェアの分析を行った研究が行われている．組込みシステム向けマルウェア Mirai の攻撃性能評価 [11] では，Mirai を VM 上で動作させ通信の様子や攻撃の流れの動作を確認し，攻撃性能を計測した．実機実験として，ローカルネットワーク上で複数の組み込みボード (odroid-c2) を用いて VM と同様に Mirai の動作環境を構築し，攻撃性能の影響の調査を行った．組み込みシステム向け TCP/IP スタックからなる http サーバーが動作する静的な組込みシステムのプロトタイプを対象に攻撃を行い DDoS 攻撃時には，CPU 使用率が大幅に上がることを明らかにした．しかし，研究結果からマルウェアに対して具体的な検知手法の提案がなされていない．IoT マルウェアによる DDos 攻撃の動的解析による観測と分析 [12] では，ハニーポットを用いて収集した IoT マルウェアの検体を用いて ARM，MIPS，MIPSEL の 3 種類の CPU アーキテクチャを用いてマルウェアを動作させその挙動を観測し，ダミー C&C サーバーを用いて攻撃再現実験を行い DoS 攻撃の観測を行った．マルウェアに対して DoS 攻撃命令が届くタイミングは各感染ホストによって様々であり，マルウェアの動作直後に集中されるわけではないことがわかった．このことから，動的解析により DoS 攻撃の命令の挙動を観測する場合には，長期的な観測が必要になる．

2.1.2　動的解析を行ったマルウェア検知の研究

マルウェアの検知手法として，API を特徴として用いた研究が広く行われている．API 呼び出しとそれに伴う経過時間とシステム負荷を用いた検知手法 [13] では，API 呼び出しパターン，API 呼び出しによる経過時間とシステム負荷を特徴量としたマルウェア検知手法を提案した．マルウェア 1 検体あたりに 10 秒間動作をさせ，その間に得られた動的解析ログから API 呼び出しとそれに伴う経過時間とメモリ使用量の情報を抽出し，マルウェアの特徴抽出を行い，機械学習アルゴリズムを用いてマルウェア検知を行う．結果として，API 遷移がほとんど重複していないマルウェアに関しては高い精度で検知を行う事ができた．しかし，呼び出される API がある程度重複しているマルウェア検体を用いた実験を行っていないため，呼び出される API が重複してない場合のマルウェアの検知精度は明らかにされていない．実行毎の挙動の差異に基づくマルウェア検知手法 [14] では，マルウェアを複数回実行した際の挙動の差異を判断することによってマルウェアの検知を行う．検査対象である 1 つのマルウェアに対して 2 回動的解析を行い，それぞれの実行時の API 呼び出しログを取得しログから特定の API の引数を抽出し 2 つの実行ログから取得した

引数が異なっている場合にマルウェアと判断した．しかし，毎回決まった動作を行うマルウェアは挙動の変動が見られないため検知ができなかった．しかしそのようなマルウェアに対してはパターンマッチング方による検知が有効だと考えられ，提案手法と組み合わせた効率的な検知手法の提案が課題になっている．この検知手法では，特定のサーバーにマルウェアだと思わしきバイナリファイルを送信し実行してログを取得しているため，Mirai のように実行後に自身のバイナリファイルを消してしまうマルウェアは動作ログを取得できないためマルウェアの検出が行えない．アノマリ手法を用いた IoT 機器マルウェア感染検出 [15] では，IoT デバイスを模したハニーポットを用いて多くのマルウェアからダウンロードされたバイナリファイル，スクリプトファイルの収集を行った．収集したファイルを用いて動的解析を行い，マルウェアが行う通信を記録した．マルウェアが行う通信が IoT デバイスの本来の通信とは異なることを明らかにし，C&C サーバーとの通信を検知することによってマルウェアの検知を行った．しかし，C&C サーバーとの通信が遮断されていたり，通信が暗号化されている場合には検知ができない．通信以外の挙動を併せて検知することでこの問題は解決可能だと考えられ，マルウェアの多様な挙動が観察可能という点で IoT デバイス上でのマルウェアの検知は妥当だと考えられる．

2.2 関連技術

2.2.1 Arbor Networks Peakflow

Arbor Networks Peakflow は Arbor Networks 社のサービスであり，Web サイトに攻撃トラフィックが届く前に DDoS 攻撃を止めてしまう DDoS 攻撃対策ソリューションとなっている [16]．図 2.1 のようにトラフィック管理技術の NetFlow などを使用してネットワーク全体をモニタリングし，DDoS 攻撃の恐れがあるトラフィックを検知する．疑わしいトラフィックにてついては，DDoS 攻撃を緩和せるスレッドマネジメントシステム (TMS) と呼ばれるに端末に中継させ正規のトラフィックだけを通信させる．このシステムでは DDoS 攻撃だと思われるトラフィックを検知してから 30 秒以内に DDoS 攻撃の緩和動作を始める．

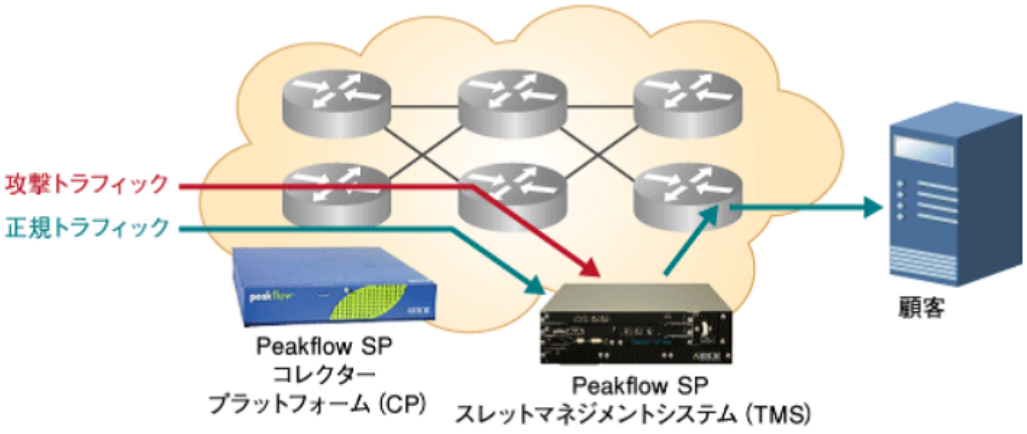


図 2.1: Arbor Networks Peakflow のシステム概要 [17]

2.2.2 Clam AntiVirus

Clam AntiVirus[18] はオープンソースで提供されているクロスプラットフォームのアンチウィルスソフトウェアである．シグネチャと呼ばれるマルウェアの特徴を記載したファイルによるパターンマッチング方式を採用しており，約 21755 種類のウィルスに対応をしている．公開されているシグネチャを用いてホスト上にあるファイルをスキャンしシグネチャと一致したファイルが無いか探索を行う．シグネチャと一致するファイルが有った場合には，ユーザーに対して通知を行う．

第3章　シンボルテーブルを用いた検知手法の提案

3.1　検知手法へのアプローチ

検知手法を定めるために，IoT デバイス上で普段の動作とマルウェアがダウンロードされ実行されたあとの動作の違いを明らかにする．その後，Mirai を実際に動作をさせデバイス上で行われている動作の解析を行う．実行コマンド，プロセスの 2 つのログデータの収集を行い，マルウェアが実行される前と後の相違点を明らかにする．

3.2　Mirai ソースコードを用いた稼働調査

Mirai の挙動の稼働調査として Web 上で公開されている Mirai のソースコード [4] を用いてマルウェアが感染する際の感染動作と C&C サーバーとの通信が行われ，攻撃命令を待機するまでのマルウェアの動作を確認した．Mirai が IoT デバイスに感染する様子を確認するために用いた，VM を用いた解析環境を図 3.1 に示す．Mirai をダウンロードさせ実行させるための感染端末，Loader，C&C サーバー，攻撃対象の管理データベースの 4 つを用意した．データベースに C&C サーバーの管理ユーザを登録し C&C サーバーと感染端末の通信状態を確認できるようにした．Loader が感染端末に Telnet ログインを行い，感染端末の通信が確立される．ログイン後に実行されるコマンドの収集を行い、表 5.1 に示すコマンド列を得た．表 5.1 のように Mirai はバイナリファイルをダウンロードした際に，バイナリファイルの名称を dvrHelper に変更している．しかし，バイナリファイルの実行後に，ps コマンドでプロセス名を確認すると，無作為なプロセス名で動作し，他の端末から Telnet ログインが不可になっていることが確認された．Mirai には，DDoS 攻撃を行う機能だけではなく，特定のポートを閉じる機能やプロセス名を無作為にする機能が存在することが確認された．

表 3.1: マルウェアによる実行コマンド

/bin/busybox wget;
/bin/busybox wget
http://192.168.32.10:80/bins/mirai.x86
-O ->dvrHelper;
/bin/busybox chmod 777 dvrHelper;
./dvrHelper telnet．x86;

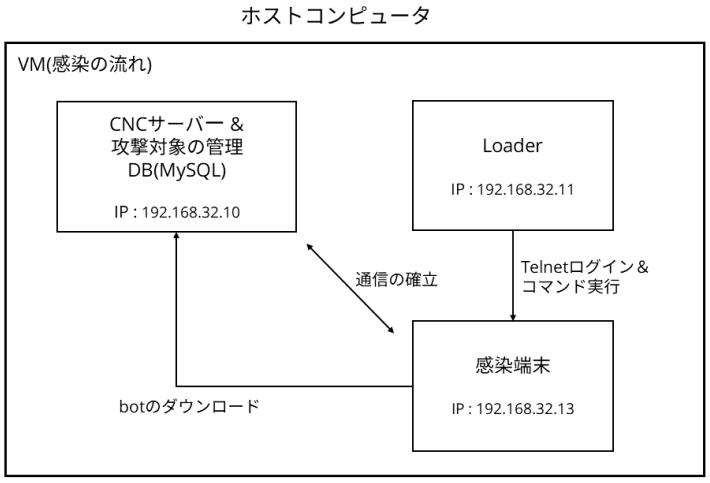


図 3.1: Mirai の解析環境

3.3 シンボルテーブルを用いた検知手法の提案

計算資源が潤沢でない IoT デバイス上でも実現可能な，Mirai 亜種の動作を検知する軽量な表層解析に基づく検知システムを提案する．検知システムの概要を図 3.2 に示す．Mirai とその亜種である Owari を含めて調査したところ，DDoS 攻撃を行うマルウェアについて亜種を含めて同様の機能を持つ，同一のコードが再利用されていることが確認された．そこで，マルウェアが持つ特定の関数の具備を検知条件として定め，この条件を満たすプロセスの稼働を検出することによってマルウェア感染の有無を判定する手法を以下に述べる．マルウェアが持つ特定の関数を具備したプログラムを検出するために，シンボルテーブルを用いる．シンボルテーブルとは，プログラム内で定義されている変数名，関数名に関する情報が保持されたデータリストである．シンボルテーブルを取得することによってプログラム内で使用されている関数名の把握が可能であることからマルウェアが持つ特定の関数を具備したプログラムを検出する．

1. IoT デバイス上で動作を行うプロセスのホワイトリストを作成する．ホワイトリストとは，端末上で可動が許可されたプロセスリストのことである．

2. プロセスを監視し、作成されたホワイトリストをもとに記載がないプロセスを発見

する．

3. ホワイトリストにないプロセスに関して，プロセスを動かしているバイナリファイルのシンボルテーブルを取得し，プロセス名を無作為に変更するなどのマルウェアの特定の関数が存在しているか確認を行う．

4. マルウェアが持つ特定の関数の存在が確認できた場合には，マルウェアだと判断を行う．

5. ホワイトリストにないプロセスに関して，シンボルテーブルの探索が終わった場合には 2 に戻る

プログラムからシンボルテーブルを取得する際にホワイトリストを用いる事によって，シンボルテーブルを取得するプログラムを限定する．それにより，シンボルテーブルを取得する回数を減らすことが可能でありシンボルテーブルを取得する際の計算資源を節約することができる．検知項目のマルウェアが持つ特定の関数として，DDoS 攻撃を行う関数や，事前調査で把握した，プロセス名を無作為にする関数や，特定のポートを閉じる関数などが候補に挙げられる．

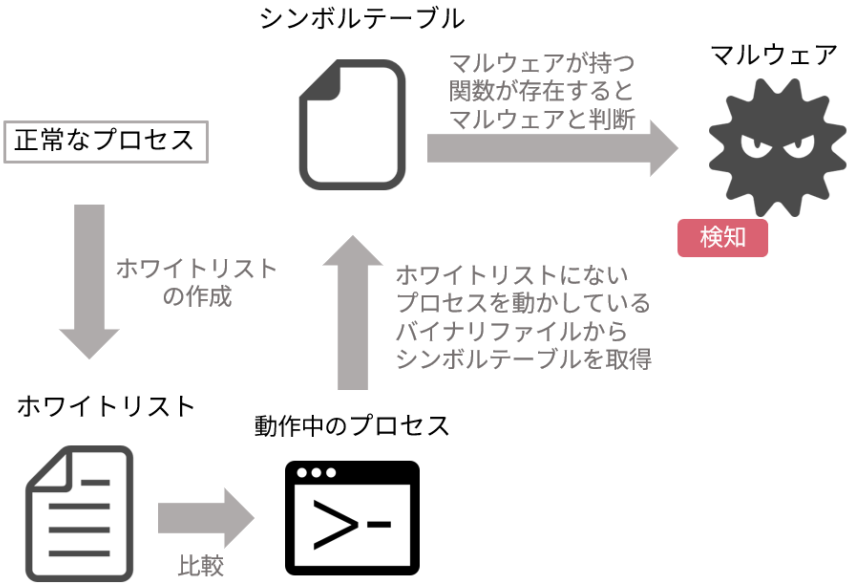


図 3.2: シンボルテーブルを用いた検知システムの概要

3.4 マルウェア探索動作による負荷と検知における制約事項

IoT デバイスは放置されることが多く，常に操作を行っているわけではない．そのため継続的にアンチウィルスソフトウェアなどの検知システムを利用して IoT デバイスにマルウェアがダウンロードされ実行されていないか確認し，IoT デバイスが安全な状態であることを把握する必要がある。検知システムのマルウェア探索動作によって，IoT デバイスの動作が妨げられる可能性がある．IoT デバイス上で Mirai など DDoS 攻撃をおこなうマルウェアが動作している際には，特定のサーバーに対して DDoS 攻撃を行ってしまうため IoT デバイスの正常な動作を妨げてまでマルウェアを検知する必要がある．しかし，IoT デバイス上でマルウェアが動作していない状況下において映像，音声，ログなどの様々なデータを伝達するなどの IoT デバイスの本来の動作がマルウェアの探索動作によって阻害されてはならない．そのため，IoT デバイスにマルウェアが動作していない状況下において，提案した検知システムによるマルウェア探索動作が IoT デバイス本来の動作を阻害していないか評価を行う．そして，IoT デバイスで Mirai が動作した場合に，提案した検知システムによって検知が可能であることを評価する．

提案手法の稼働に必要なマルウェア探索動作によって IoT デバイス本来の動作が阻害されてないことを評価するために，LinuxOS を対象とする既存のアンチウィルスソフトである Clam AntiVirus(ClamAV) を動作させた状態の CPU とメモリ使用率をそれぞれ基準値とし，提案手法によるマルウェア探索動作の動作負荷について比較を行い，併せて提案手法によって Mirai マルウェアの検知が可能であることを確認した．提案した検知システムによって Mirai が動作していない状況での、CPU，メモリの使用率について調査した．システムの負荷状況を確認する sar コマンドを用いて 1 分間計測を行なった．表 3.2 の性能のラズベリーパイを利用し負荷状況を測定した．

表 3.2: 評価環境の IoT デバイスのスペック

Raspberry Pi3	
OS	Openwrt 4.9.120
CPU	Quad Core 1.2GHz Broadcom BCM2837
MEM	1GB

sar コマンドを用いて得た CPU，メモリの使用率について Clam AntiVirus と提案した検知システムの比較を行った結果が図 3.3，3.4 になる．Clam AntiVirus を利用した場合には，平均 CPU 使用率が 25.28%，メモリ使用率は 7.93%となった．提案した検知手法では，平均 CPU 使用率が 3.03%，メモリ使用率が 7.21%となった．メモリ使用率は比較対象の Clam AntiVirus と提案した検知手法では 12.5%減，CPU 使用率は，Clam AntiVirus に対して提案した検知手法は 88%減となったことからマルウェアの可動を検知する目的で一般的によく利用される Clam AntiVirus に比較して提案手法の実装では資源消費が少なく他のプロセスの動作を妨げる可能性は低いと言える．しかし，提案した検知した検知手法は実行形式ファイルに含まれるシンボルテーブルの内容に基づいている為，マルウェアの実行形式ファイルに対して strip コマンドを用いるなどしてシンボルテーブルが削除された場合には検知が行えないという課題がある．

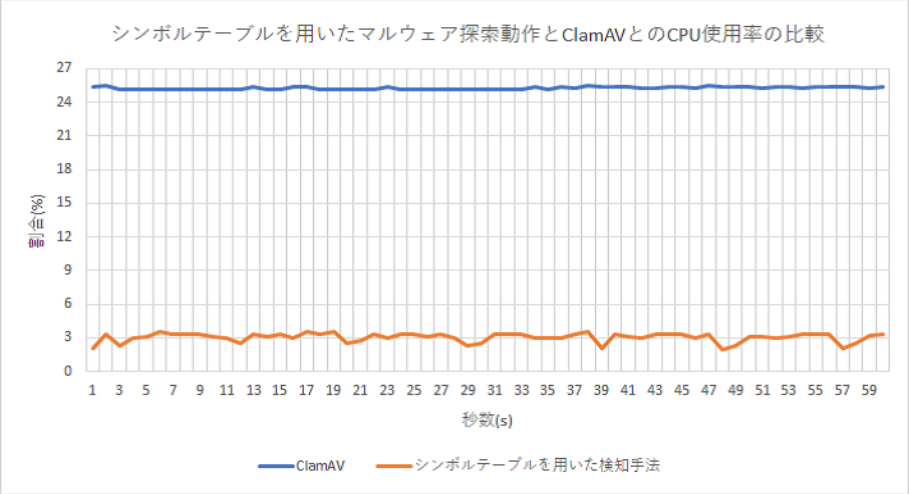


図 3.3: シンボルテーブルを用いたマルウェア探索動作と ClamAV との CPU 使用率の比較

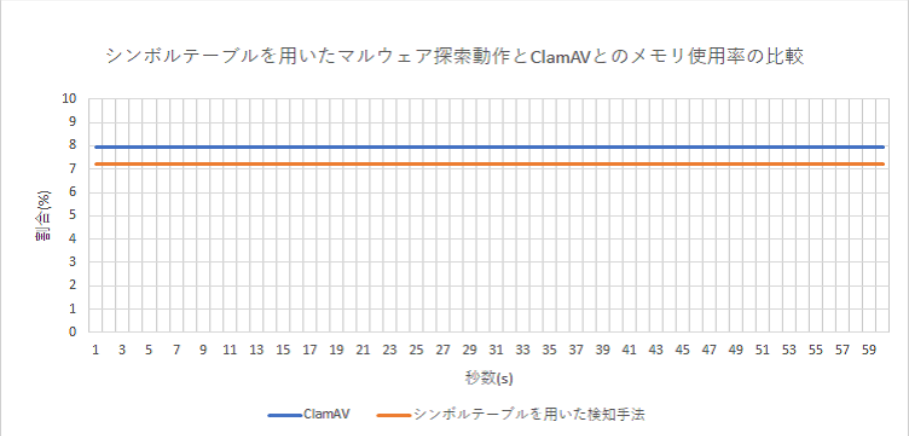


図 3.4: シンボルテーブルを用いたマルウェア探索動作と ClamAV とのメモリ使用率の比較

第4章　システムコール呼び出し履歴を用いた検知手法の提案

4.1　新たな検知手法の必要性

前章で述べたシンボルテーブルを用いてマルウェアの検知を行う検知手法では，strip コマンドを用いてシンボルテーブルを削除したり，検知条件となっている関数名を変更するといった攻撃者側による検知回避の対処が取られた際には有効な検知が行えないという課題がある．関数が呼び出すシステムコールの呼び出し系列は関数名の名称を変更しただけでは変化しない．動作しているプロセスから呼び出されているシステムコールを追跡し，Mirai マルウェアのプログラムにおいて特徴的な動作を実装した内部関数に着目しこの関数から呼び出されるシステムコールの系列を用いた検知を行うことによって検知回避の対処がなされた場合でも検知が可能になる．

4.2　Mirai の特徴的な動作に基づく検知条件

Mirai マルウェアは特徴的な動作として，サーバーに DDoS 攻撃を行う動作の他にインターネットに公開されているホストに対して新たな侵入先を見つけるために telnet ログインが可能な端末をスキャンする活動を行っている．Mirai はサーバーに DDoS 攻撃を行うプロセスと telnet ログインが可能な端末をスキャンする活動のプロセスは独立して動作しているため，プロセスは別々に存在している．DDoS 攻撃を行うプロセスは DoS 攻撃を行っている場合や，攻撃命令を待機している場合など動的解析を行うタイミングによって，異なった解析結果が得られることが考えられる．しかし，スキャン活動を行うプロセスはログインできる端末を常時探索しているため動的解析を任意のタイミングで行っても，一定の解析結果が得られると考えられる．プロセスによって呼び出されているシステムコールを追跡するために strace コマンドを利用した．DDoS 攻撃を行うためのプロセスとスキャン活動を行っているプロセスについて，それぞれシステムコールを追跡したところ，DoS 攻撃を行うプロセスは攻撃命令を待機している状態になるまでに呼び出されるシステムコールは様々なものがあった．しかし，スキャン活動を行っているプロセスはソケットへメッセージを送るシステムコールである sendto を連続して呼び出していた．そのため，任意のタイミングでシステムコールを追跡しても同一の結果が期待できる．そのため，スキャン活動を行うプロセスに着目をしスキャン活動が呼び出すシステムコールの系列を用いた検知を行う．スキャン活動を行うプロセスのシステムコールの実行状況を追跡したところ sendto を連続して呼び出しており，sendto によって送信されるメッセージの宛先アドレスが呼び出しごとに異なったアドレスであること，送信先のポートが 23 であったことからこのシステムコールを検知に用いる特徴とする．検知条件として 3 つの条件を定める．

1. sendto のシステムコールが 2 回以上連続して呼び出されていること
2. sendto 呼び出しに指定された送信先のポートが 23 であること
3. sendto によって送信されるメッセージの宛先アドレスが呼び出しごとに異なったアドレスであること

4.3 誤検知の可能性

前章で述べた検知条件をもとにマルウェア探索を行った際に，誤検知する場合として，以下の 3 つが考えられる

1. IoT デバイス上で sendto の呼び出しが多いプログラムの実行
2. IoT デバイスから複数の端末に向けてメッセージを送信
3. IoT デバイスから複数の端末を遠隔操作しサーバー等の設定やログファイルを特定のサーバーへ転送

strace を用いてシステムコールを確認し上記の動作が検知条件に一致するのか確認を行った．1 の紛らわしいプログラムの例として，IoT デバイス上で sendto のみを一定時間繰り返し呼び出すプログラムについて考える．sendto が呼び出されるだけのプログラムでは，検知条件に一致しやすく誤検知する可能性がある．しかし，送信先のポートが 23 であり，送信されるメッセージの宛先がすべて別の宛先アドレスである sendto が呼び続ける正規プログラムが存在するとは考えにくい．

2 の IoT デバイスから複数の端末に向けてメッセージを送る動作として考えられるものが，wall や write など IoT デバイスに telnet，ssh ログインしている端末にメッセージを送るコマンドがある．wall，write コマンドを実行してシステムコールを確認した結果が表 4.1 のようになる．表 4.1 のようにメッセージを他の端末に送信する際，sendto を呼び

表 4.1: メッセージの送信時に呼び出されるシステムコール

wall	write
read(0,”Hello World\n”,1024)	read(0,”Hello World\n”,1024) write(1,”Hello World\r\n”)
read(0,”test\n”,1024)	read(0,”test\n”,1024) write(1,”test\r\n”)
read(0,”Mirai\n”,1024)	read(0,”Mirai\n”,1024) write(1,”Mirai\r\n”)
read(0,”Good-bye\n”,1024)	read(0,”Good-bye\n”,1024) write(1,”Good-bye\r\n”)

出すことが確認されなかったため，IoT デバイスに telnet，ssh ログインを行っている複数の端末に向けてメッセージを送信する場合には誤検知することがない．

3 の IoT デバイスから複数の端末を遠隔操作する方法について，ssh や telnet，parallel-ssh といったリモートシェルを用いて手動でコマンドを入力して端末を操作する場合とスクリプトファイルなどで端末を自動的に操作させる 2 種類がある．IoT デバイスから複数端末に手動でコマンドを入力してファイルの転送などを行いシステムコールを確認したところ，連続の sendto 呼び出しは見られなかった．スクリプトファイルを利用してファイルの転送を行う場合も，同様に sendto を連続で呼び出していることを確認できなかった．sendto だけを呼び出すプログラムを telnet，ssh を使用して端末上で実行した場合，select のシステムコールが呼び出され sendto が 2 回以上連続で呼び出されていることが確認できなかった．ssh や telnet を利用して遠隔操作を行う場合や他の端末にメッセージを送信する場合には sendto が 2 回以上連続で呼び出されていることがないため誤検知することはないと考えられるため，これらの検知条件は妥当だと考える．

4.4 strace コマンドによる監視対象のプロセスの実行速度の調査

strace コマンドによってシステムコール呼び出し履歴を監視されるプロセスの実行速度が低下することが考えられる．そのため，strace コマンドによってプログラムの実行速度の変化を調べ strace によるシステムコールの監視動作がプロセスに与える実行速度の影響を調査した．使用するプログラムとしては，農林水産研究情報総合センターが定める，よく使用される Linux コマンド [19] を使用した．cp，tar，df，ps，cat の 5 つのコマンドを対象に strace によるシステムコールの監視動作によるプロセスの実行速度の影響を調査した．実行したコマンドはは表 4.2 になる．

表 4.2: 実行したコマンド

cat gpl-2.0.txt
cp gpl-2.0.txt cp.txt
ps -w
df
tar -zcvf gpl-2.0.tar.gz gpl-2.0.txt

コマンドの引数にファイルが必要になる場合にはフリーソフトライセンスである GPL2 の内容が書かれているテキストファイルを使用した．表 4.2 の 5 つコマンドに対しての strace コマンドを実行した場合の速度比較を行った．実行した結果が表 4.3 になる．

表 4.3: 計測結果

	無為 (秒)	strace(秒)
cat	3.5730	3.6894
cp	0.0048	0.2517
ps	0.5134	10.9312
df	0.6264	0.1212
tar	0.57014	0.0294

システムコールの監視動作による速度は cat コマンドは 3%低下，cp コマンドは 5079%低下，ps コマンドは 2029%低下，df コマンドは 417%低下，tar コマンドは 1839%低下．この結果から，呼び出されるシステムコールによって速度の低下量は変化するがシステムコール監視動作によってプロセスの実行速度は低下することがわかった．システムコール監視対象となるプロセスが正常なプロセスだった場合，実行速度の低下がするため，マルウェア探索動作による 1 プロセスあたりの strace の実行回数，アタッチする時間を短くする必要がある．

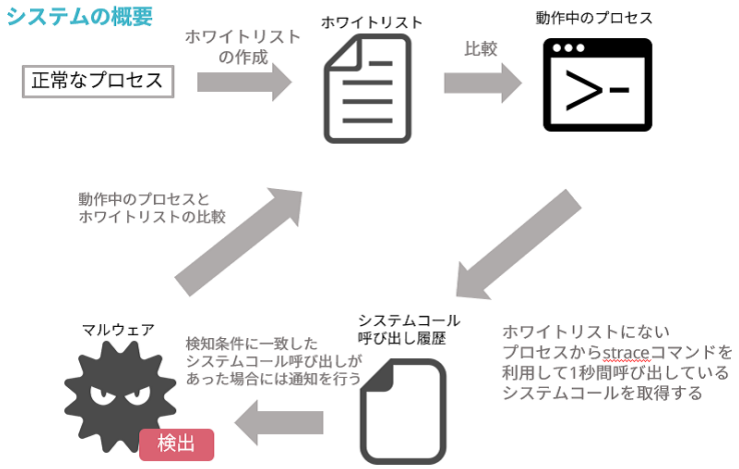


図 4.1: システムコール呼び出し履歴を用いた検知システムの概要

4.5　システムコール呼び出し履歴を用いた検知手法の提案

計算資源が潤沢でない IoT デバイス上でも実現可能な，Mirai 亜種の動作を検知する軽量な動的解析に基づく検知システムを提案する．検知システムの概要を図 4.1 に示す．Mirai は telnet ログインが可能な端末を探索するスキャン活動を行う機能を持ち，システムコールの一種である sendto を複数回連続で呼び出している．そこで動作しているプロセスからシステムコール呼び出し履歴を取得し，スキャン活動を行っているプロセスの動作を確認することでマルウェア感染の有無を判定する手法を以下に述べる．

1. IoT デバイス上で動作を行うプロセスのホワイトリストを作成する．
2. プロセスを監視し、作成されたホワイトリストをもとに記載がないプロセスを特定する．
3. 特定した複数のプロセスに関して，strace を 1 秒間実行し検知条件に一致したシステムコール呼び出し呼び出し履歴があるか監視する．もし検知条件に一致したシステムコール呼び出し履歴があった場合にはマルウェアだと判断を行い通知を行う．
4. 1 プロセスあたりの strace によるシステムコール監視動作の回数を減らすために，strace を実行したあとに監視動を中断する時間を設ける．2.2.1 項で述べた Arbor

Networks Peakflow は DDoS 攻撃を検知してから DDoS 攻撃の緩和動作を 30 秒以内に行う．これを参考値とし，マルウェアを検出するのに要する時間の最悪値を 30 秒とする．監視動作を中断する時間は以下の式で求める．

$$\text{中断時間} = \frac{\text{最悪値}}{\text{監視プロセス数}} - 1 \tag{4.1}$$

5. 複数の監視対象となるプロセスに対して strace を実行する．すべての監視対象となるプロセスに対して strace を実行した場合には 2 に戻り繰り返す．

検知システムとして前章で述べたシンボルテーブルを用いた検知システムに変更したものを利用した．

第5章　システムコール呼び出し履歴を用いた検知手法による評価実験

本研究で実装した検知システムにおける定常的な動作負荷の計測を行い，IoT デバイスが本来の動作を阻害しないことを評価する．DDoS 攻撃を行うマルウェアを用いて検知精度の評価を行い，システムコール呼び出し履歴による検知手法でマルウェアが検出できることを確認した．

5.1　システムコール呼び出し履歴を用いた検知手法による定常的な動作負荷の評価

IoT デバイスにマルウェアが動作していない状況下において，提案した検知システムによるマルウェア探索動作によって IoT デバイス本来の動作が阻害されていないことを評価するために，3.4 節で述べたように，LinuxOS を対象とする既存のアンチウィルスソフトである Clam AV を動作させた状態の CPU，メモリの使用率を基準値とし，提案した検知システムによって Mirai が動作していない状況での，CPU，メモリの使用率について sar コマンドを用いて 1 分間計測を行った．sar コマンドを用いて得た CPU，メモリの使用率について，Clam AV と比較を行った結果が図 5.1，5.2 になる．ClamAV を動作させた状態の IoT デバイスの CPU 使用率，メモリ使用率の値は 3.4 節で計測したデータを用いた．Clam AntiVirus を利用した場合には，平均 CPU 使用率が 24．27%，メモリ使用率は 7.92%であり．提案した検知手法では，平均 CPU 使用率が 0.57%，メモリ使用率が 4.08%となった．メモリ使用率は比較対象の Clam AntiVirus と提案した検知手法では 97.6%減，CPU 使用率は，Clam AntiVirus に対して提案した検知手法は 48.4%減となった．

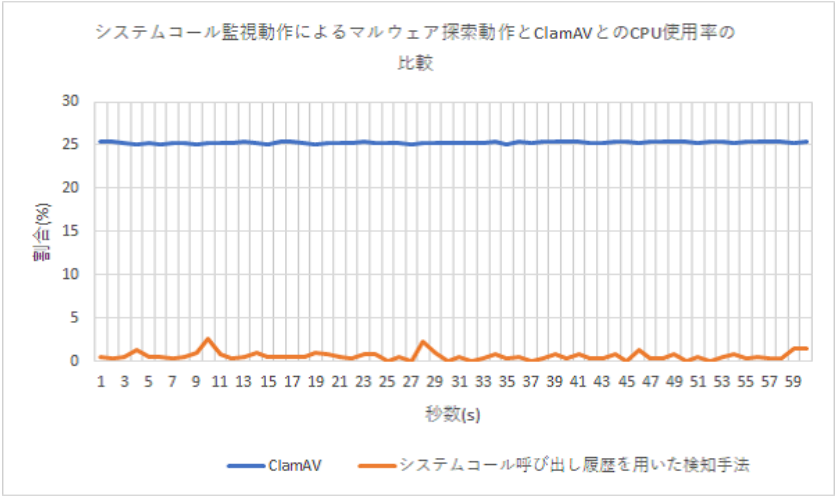


図 5.1: システムコール呼び出し履歴によるマルウェア探索動作と ClamAV との CPU 使用率の比較

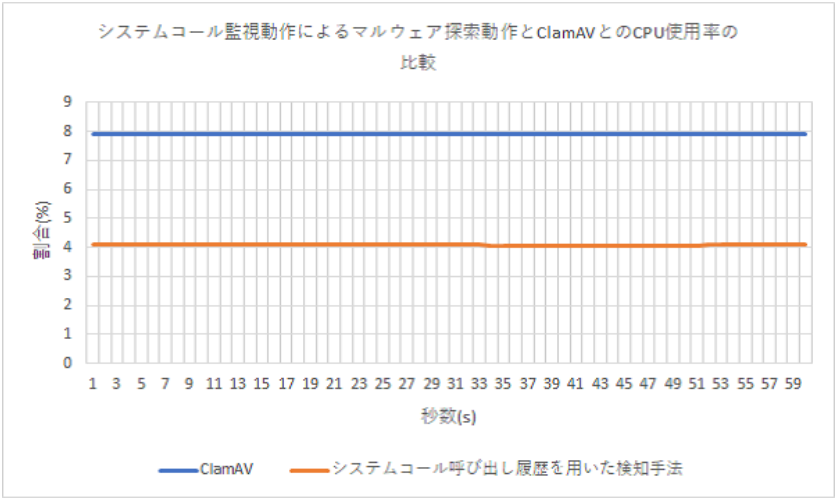


図 5.2: IoT デバイス上でマルウェアが動作していない状況におけるマルウェア探索のメモリ使用率

5.2 Mirai とその亜種マルウェアを対象とする判別性能評価

ハニーポットを用いて DDoS 攻撃を行うマルウェアを収集し，収集したマルウェアを検体として用いて，システムコール呼び出し履歴を用いた検知手法の検知精度の評価を行い，システムコール呼び出し履歴を用いた検知手法の有効性を確認した．

5.2.1 ハニーポットによるマルウェアの収集

ハニーポットと呼ばれる攻撃者に脆弱なシステムであると見せかけることで攻撃者を誘い込み，侵入手法や侵入後に実行されるコマンドのログやダウンロードされるファイルを収集するシステムを用いて DDoS 攻撃を行うマルウェアを収集した．ハニーポットのシステムの概要図を図 5.3 に示す．シェルの対話の中でダウンロードされるバイナリファイルを実行させることなく保存することが可能な Michel Oosterhof によって開発された Cowrie[20] と呼ばれるハニーポットを用いた．Cowrie によって収集されたバイナリファイルについて Virus Total と呼ばれるマルウェア検知オンラインサービスを用いて解析を行い，DDoS 攻撃を行うマルウェアの分類を行った．Virus Total[21] はユーザーから投稿された検体を 54 のウィルス対策エンジンによって解析するオンラインサービスであり，投稿された検体についてマルウェアの分類を知ることができる．2019/01/09 から 2019/01/28 の期間でハニーポットを断続的に運用してバイナリファイルの収集を行った．収集したバイナリファイルを Virus Total に投稿し，Virus Total の解析結果から，Mirai または Mirai の亜種のマルウェアを DDoS 攻撃を行うマルウェアとして分類分けした．分析した結果，収集したバイナリファイルは，空ファイルのものや，マルウェアをダウンロードさせ実行させるファイル，DDoS 攻撃を行うマルウェアのバイナリファイル等が散見され，DDoS 攻撃を行うマルウェアは 57 検体が存在し，マルウェアの実行可能な検体は 49 検体であった．評価に用いたマルウェアの検体の情報は付録 A に記載する．

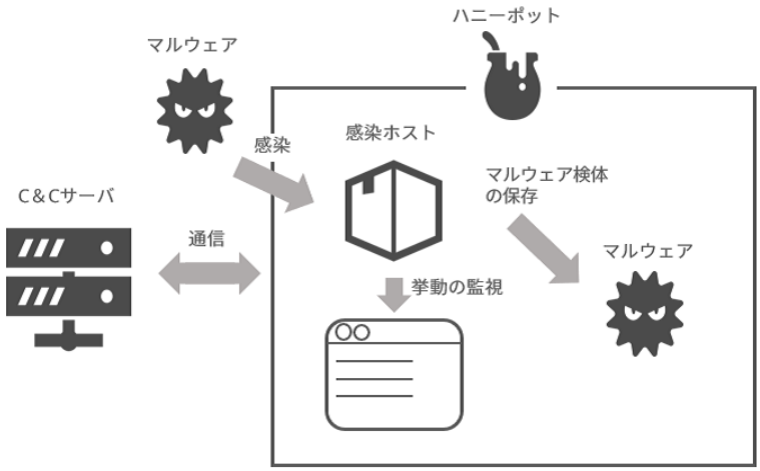


図 5.3: ハニーポットのシステム概要図

5.2.2 提案手法によるマルウェアの検知精度評価

前項で収集したDDoS攻撃を行うマルウェア49検体を用いてシステムコール呼び出し履歴を用いた検知手法の検知精度の評価を行った．入手したマルウェアをネットワークから隔離した状態で検知システムを動作させマルウェアの検出ができるか評価を行った．正規プログラムは，Linuxであるubuntu 16.0 LTS4 LTSに標準でインストールされているプログラムの44検体を利用した．実験の評価指数として，Accuracy，TPR(True Positive Rate)，TNR(True Negative Rate)，FPR(False Positive Rate)，FNR(False Negative Rate)を用いる．Accuracyは，マルウェアを正しく判別できた割合である．TPRはマルウェアをマルウェアと判別できた割合，TNRは正規プログラムを正規プログラムと判別した割合，FNRはマルウェアを正規プログラムと判別した割合，FPRは正規プログラムをマルウェアとして判別した割合である．Accuracy，TPR，TNR，FPR，FNRを下記の式で算出する．

$$Accuracy = \frac{\text{正しくマルウェアと判別された検体数}}{\text{検体の総数}} \tag{5.1}$$

$$TPR = \frac{\text{マルウェア検体がマルウェアと判別された検体数}}{\text{マルウェア検体の総数}} \tag{5.2}$$

$$TNR = \frac{\text{正規プログラムを正規プログラムとして判別された検体数}}{\text{正規プログラムの総数}} \tag{5.3}$$

$$FNR = \frac{\text{マルウェア検体が正規プログラムとして判別された検体数}}{\text{マルウェア検体の総数}} \tag{5.4}$$

$$FPR = \frac{\text{正規プログラムをマルウェアとして判別された検体数}}{\text{正規プログラムの総数}} \tag{5.5}$$

提案システムによるハニーポットで収集したマルウェアの判別結果を表5.1に示す．プログラムの判別結果を表5.2に示し，Accuracy，TPR，TNR，FPR，FNRの結果を表5.3に示す．

表 5.1: 49検体のマルウェアの種類と検知の可否

検体番号	マルウェアの種類	検知の可否	検知できない理由
1	Linux.Mirai.793	可能	-
2	Linux.Mirai.793	可能	-
3	Linux/DDoS-Xor.A	可能	-
4	Linux/Mirai	可能	-
5	Linux/Mirai	可能	-
6	Linux/Mirai	可能	-
7	Linux/mirai.d	可能	-
8	Linux/mirai.d	可能	-
9	Linux/mirai.d	可能	-
10	Linux/mirai.d	可能	-
11	Linux/mirai.d	可能	-
12	Linux/mirai.d	可能	-
13	Linux/Mirai.f	可能	-
14	Linux/Mirai.f	可能	-
15	Linux/Mirai.f	可能	-
16	Linux/Mirai.f	可能	-

表 5.1: 49 検体のマルウェアの種類と検知の可否

検体番号	マルウェアの種類	検知の可否	検知できない理由
17	Linux/Mirai.f	可能	-
18	Linux/Mirai.g	可能	-
19	Linux/Mirai.g	不可	sendto 呼び出しに指定された送信先のポートが 52869
20	Linux/Mirai.l	可能	-
21	Linux/Mirai.l	可能	-
22	Linux/Mirai.l	可能	-
23	Linux/Mirai.l	可能	-
24	Linux/Mirai.l	可能	-
25	Linux/Mirai.l	可能	-
26	Linux/Mirai.l	可能	-
27	Linux/Mirai.l	不可	呼び出されているシステムコールが sendto ではなく send
28	Linux/Mirai.l	不可	sendto 呼び出しに指定された送信先のポートが 80
29	RDN/Generic BackDoor	不可	スキャン活動を行っていない
30	RDN/Generic BackDoor	不可	
31	RDN/Generic BackDoor	可能	-
32	RDN/Generic BackDoor	可能	-
33	RDN/Generic BackDoor	可能	-
34	RDN/Generic BackDoor	可能	-
35	RDN/Generic BackDoor	可能	-
36	RDN/Generic BackDoor	不可	呼び出されているシステムコールが sendto ではなく send
37	RDN/Generic BackDoor	可能	-
38	RDN/Generic BackDoor	可能	-
39	RDN/Generic BackDoor	可能	-
40	RDN/Generic BackDoor	可能	-
41	RDN/Generic BackDoor	可能	-
42	RDN/Generic BackDoor	可能	-
43	RDN/Generic BackDoor	可能	-
44	RDN/Generic BackDoor	可能	-
45	RDN/Generic BackDoor	不可	sendto 呼び出しに指定されたポートが 37215
46	RDN/Generic BackDoor	可能	-
47	RDN/Generic BackDoor	可能	-
48	RDN/Generic BackDoor	可能	-
49	RDN/Generic BackDoor	不可	検知プログラムの強制終了

表 5.2: 提案システムによるマルウェアの判別結果

		判別結果	
		マルウェア	正規プログラム
真の結果	マルウェア	41	8
	正規プログラム	0	44

評価の結果，Accuary は 91.3%と高い精度となっており，TPR の値が，83.4%であり，FPR の値が 0%である．TPR が 83.4%のため，Mirai 亜種の多くのマルウェアは作成時に元にした Mirai のスキャン機能をそのまま流用している場合が多いことがわかった．しかし，スキャン活動を行っている一部のマルウェアは提案システムによって検出することが

表 5.3: プログラムの判別結果

Accuracy	TPR	TNR	FNR	FPR
91.3%	83.4%	100.0%	16.6%	0.0%

できなかった．検出ができなかった要因としてスキャン活動を行っていない場合とスキャン活動を行っている際にスキャンしているポートが 23 ではなく，他のポートをスキャンしてため，検知条件に一致しなかったためである．TNR が高い精度を示した理由には，対象となっている正規プログラムが ubuntu 16.04 LTS に標準でインストールされているプログラムを利用している事が挙げられる．検知条件に一致する組織や個人で作成されたプログラムでは sendto を連続して呼び出すプログラムが存在すると考えられるため誤検知をする可能性がある．

5.2.3 考察

システムコール呼び出し履歴を用いた検知手法と ClamAV のメモリ使用率，CPU 使用率の比較を行った結果，メモリ使用率が 48.4%減，CPU 使用率が 97.6%減となったことから，システムコール呼び出し履歴を用いた検知手法は IoT デバイスの本来の動作を阻害することがなく IoT デバイスがマルウェアに感染していないことを確認する事ができる．Accuary は 91.3%と高い精度となっており，DDoS 攻撃を行うマルウェアを検出するのに有効である．したがって，マルウェアの内部関数によって呼び出されるシステムコール系列をもとにマルウェアを検出することによって，IoT デバイスなどの計算資源の乏しい端末でもホスト上でのマルウェアに対して有効なセキュリティ対策が行えると考えられる．

第6章　結論

6.1　まとめ

本研究では，IoT デバイス上でマルウェアを検出する手法として Mirai が持つ関数に着目し，シンボルテーブルを用いた検知手法，システムコール呼び出し履歴を用いた検知手法の 2 つを提案した．ClamAV と実装したシンボルテーブルを用いた検知システムの比較を行ったところ，メモリ使用率は 12.5%減，CPU 使用率は 88%減であった．シンボルテーブルを用いた検知手法では，マルウェアの実行形式ファイルに含まれるシンボルテーブルを削除された場合にはマルウェアの検出できない制約がある．実際にハニーポットで収集したマルウェアの殆どはこの対処がとられており，シンボルテーブルから使用される関数名を取得することができなかったためマルウェアの検出ができなかった。ClamAV とシステムコール呼び出し履歴を用いた検知システムの比較を行ったところ，メモリ使用率は 48．4%減，CPU 使用率は 97．6%減であった．2 つの提案手法では，Clam AntiVirus よりも IoT デバイスの少ない計算資源によってマルウェア探索を行うことができる．ハニーポットによって収集した DDoS 攻撃を行うマルウェアを用いてシステムコール呼び出し履歴を用いた検知手法のマルウェアの検知精度を評価した結果，Accuracy は 91.4%，FPR の値が 0%と高い精度を示していた．しかし，Miria のスキャンする port が変更されていた場合や検知プログラムを強制終了された場合にはマルウェアの検出をすることができなかった．

システムコール呼び出し履歴を用いた検知手法では，IoT デバイス上でマルウェアが C&C サーバーからの DoS 攻撃命令を待機している状態でも，マルウェアの検出が行えるため，IoT デバイス本体でマルウェア検出するのに有効であると考えられる．

6.2　今後の展望

今後の展望として，本提案手法では bashlite と呼ばれるスキャン活動を行っていない DDoS 攻撃を行うマルウェアは検知を行う事ができない．マルウェアのスキャン活動ではない挙動に着目をし新たな検知条件を定め上記のマルウェアも検知できるようにする必要がある．スキャン活動に着目した検知条件でも，ハニーポットを用いて収集された Mirai の亜種マルウェアはスキャンしている port が 23 だけではなく他の port をスキャンしていることが確認されたため，マルウェアの侵入経路を明らかにし様々な port に対してスキャン活動が行われた場合でも，マルウェアの検出ができるよう検知条件を定める必要がある．本提案手法による不正な挙動の検出から逃れるために，攻撃者側が IoT デバイスに Telnet ログインが成功にし，マルウェアをデバイス上にダウンロードさせる際に，kill コマンドを用いて提案システムを停止させてからマルウェアの実行，ホワイトリストの改ざ

んによって，マルウェアの挙動を提案システムによって監視することができず，マルウェアの検出ができない．しかし，検知動作の強制終了やホワイトリストが改ざんされる挙動を検出することがでできれば，スキャン活動を行っていないマルウェアでも検出することが可能であると考えられる．

謝辞

本研究を進めるに当たり，ご多忙な中，研究テーマの決定から論文や発表資料の添削など，熱心にご指導していただいた稲村浩教授，中村嘉隆准教授に深く感謝申し上げます．そして，日頃からお世話になった稲村浩研究室・中村嘉隆研究室の皆様，本研究の発表を聞いて大変有益なご指導，助言をしていただいた学生，教員の皆様に心から感謝いたします．

参考文献

[1] 総務省：IoT デバイスの急速な普及，情報通信白書（オンライン），
入手先<http://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h30/html/nd111200.html>（参照 2018-06-17）

[2] 宮田健 : 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，Tech Factry（オンライン），入手先<http://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html>（参照 2018-06-20）

[3] Scott Hilton : Dyn Analysis Summary Of Friday October 21 Attack，Oracle Dyn（オンライン），入手先<https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21-attack>（参照 2018-06-20）

[4] Jerry Gamblin : jgamblin/Mirai-Source-Code，GitHub，（オンライン），入手先<https://github.com/jgamblin/Mirai-Source-Code>（参照 2018-09-20）

[5] 鈴木聖子：IoT デバイスの脆弱性を突くマルウェア「Wicked」、Mirai の新手の亜種，ITmedia エンタープライズ,（オンライン）入手先<http://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html>
（参照 2018-06-20）

[6] @IT：マルウェア「Satori」による攻撃を国内初観測、従来のファイアウォール機能では対応が難しい？，@IT,（オンライン）入手先<http://www.atmarkit.co.jp/ait/articles/1806/27/news083.html>（参照 2018-09-23）

[7] Nick Lewis：Mirai 亜種の IoT マルウェア「Okiru」とは？ 標的は「ARC プロセッサ」，Tech Factry（オンライン），入手先<http://techfactory.itmedia.co.jp/tf/articles/1704/13/news010.html>
（参照 2018-06-20）

[8] 岩崎 宰守:IoT マルウェア「Mirai」を Windows から拡散、2017 年に入って 500 システムへの攻撃を確認 -INTERNET Watch，入手先<https://internet.watch.impress.co.jp/docs/news1046239.html>（参照 2018-012-26）

[9] 国立研究開発法人情報通信研究機構：日本国内でインターネットに接続された IoT 機器等に関する事前調査の実施について， NICT-情報通信研究機構（オンライン），入手先 <https://dyn.com/blog/dyn-analysis-summary-of-friday-october-21- attack>
（参照 2018-06-20）

[10] 青木一史，秋山満照，幾世知範 ほか：実践サイバーセキュリティモニタリング，pp103-105，コロナ社（2016）

[11] 長柄啓吾，松原豊，青木克憲 ほか：組込みシステム向けマルウェア Mirai の攻撃性能評価，研究報告システム・アーキテクチャ，vol.2017-ARC-225，No.41，p1-6（2017）

[12] 鉄穎，楊笛，松本勉 ほか：IoT マルウェアによる DDoS 攻撃の動的解析による観測と分析，情報処理学会論文誌，vol.59 No.5，p1321-1333（2018）

[13] 佐藤純子，花田真樹，面和成 ほか：API 呼び出しとそれに伴う経過時間とシステム負荷を用いた検知手法，コンピュータセキュリティシンポジウム 2017 論文集，vol.2017 No.2 （2017）

[14] 実行毎の挙動の差異に基づくマルウェア検知手法，上原哲太郎：アノマリ検知手法を用いた IoT 機器のマルウェア感染検出，研究報告セキュリティ心理学とトラスト，vol.2018-SRT-27 No.3，p1-6 （2018）

[15] 坂野加奈，上原哲太郎：アノマリ検知手法を用いた IoT 機器のマルウェア感染検出，研究報告セキュリティ心理学とトラスト，vol.2018-SRT-27 No.3，p1-6 （2018）

[16] Arbor：Arbor Networks Peakflow® 7.0 が、 DDoS 攻撃検知とミティゲーションの時間を大幅に短縮，入手先 <https://jp.arbornetworks.com/lorem-post-3/>（参照 2018-12-22）.

[17] ビジネスネットワーク：アーバーネットワークス　Arbor Peakflow SP/TMS　あらゆる企業が DDoS 攻撃で狙われる時代　No.1 ベンダーの答えは「上流で止めろ！」，入手先 <https://businessnetwork.jp/Portals/0/SP/1210_arbor/?prtext>（参照 2019-1-21）.

[18] Nicholas Herbert，Melissa Taylor，Nicolette Verbeck：ClamAV,（オンライン），入手先 <https://www.clamav.Networks>（参照 2018-12-22）.

[19] 農林水産研究情報総合センター：UNIX（Linux）でよく使うコマンド ，（オンライン），入手先 <https://itcweb.cc.affrc.go.jp/affrit/doku.php?id=faq/tips/unix-command>（参照 2019-1-14）

[20] Micheloosterhof:Cowrie,（オンライン），入手先 <https://itcweb.cc.affrc.go.jp/affrit/doku.php?id=faq/tips/unix-command>（参照 2019-1-4）

[21] Google Inc：Virus Total,（オンライン），入手先，<https://www.virustotal.com/home/upload>（参照 2019-01-14）

付 録A　マルウェア検体の情報

検体番号	SHA-256 ハッシュ値
検体 1	1f2fd1d29d1bc21cfdab81f2805c1bc87aa60f16c6c478d2bbf5a84432cf279c
検体 2	6c7c1e6aedc744985e440b8a07d5803256d88cb9df8d72d25014cbf0081a50f7
検体 3	90ad1f172af7d0915e548bd84443ab3cc3b3df97b3fbf8c06ecc8b42604fbb5f
検体 4	f919b9a88cd4aedf43145916d33f9ca10202735acec3b052b842cfdbaf5ba27b
検体 5	f919b9a88cd4aedf43145916d33f9ca10202735acec3b052b842cfdbaf5ba27b
検体 6	f919b9a88cd4aedf43145916d33f9ca10202735acec3b052b842cfdbaf5ba27b
検体 7	0fd6fce6cea829f26afdc177d62bcd15f953b4a9b9674033cf2a016ec306c09b
検体 8	15221c835abf22d1c3263f69056690c1836b0805730a90edb7caa486809c6b5d
検体 9	98493f5651389732a8413493b5d3b2c0996e396fb2906e9d06d3abd44d04cc53
検体 10	98493f5651389732a8413493b5d3b2c0996e396fb2906e9d06d3abd44d04cc53
検体 11	c0a7ef03b9dfeef1fbbc664a8119d70a9b11d7e6433f5a9441a97cb98f6d9ab0
検体 12	c192f4e4766852fb8d0e09498990a6c974527ba9a8bbb59a347bed74c7468786
検体 13	2dacf4473a5aef5b4911820b65edf7963e431a3e6dcbcbad2356775451d6631e
検体 14	e8e55bf7dab6443de1750a29b2bf91532528171565bcc6a85c46f981df6bc54c
検体 15	e8e55bf7dab6443de1750a29b2bf91532528171565bcc6a85c46f981df6bc54c
検体 16	e8e55bf7dab6443de1750a29b2bf91532528171565bcc6a85c46f981df6bc54c
検体 17	ebfe9295ea16fae87d97b9cb57cc2457cfec688a2e2f2f23965afcc4b3a58079
検体 18	727b5dcaaaab23c55382c9f818b814c71544d96656e43f82419a938c84dfdbcb
検体 19	de27ec703af00d1a5ccf27b5460c50e2c53e6bf614fd94fba30dfc7a9e6f3132
検体 20	20206452be66a7205a8150acd965b77736bb5457ad29f7208d0abb21ece31312
検体 21	20206452be66a7205a8150acd965b77736bb5457ad29f7208d0abb21ece31312
検体 22	20206452be66a7205a8150acd965b77736bb5457ad29f7208d0abb21ece31312
検体 23	34a7d31aae48f5fb3ef18ae8122d48773d81dffb75b4a2cb0c4993b9092d4f68
検体 24	34a7d31aae48f5fb3ef18ae8122d48773d81dffb75b4a2cb0c4993b9092d4f68
検体 25	34a7d31aae48f5fb3ef18ae8122d48773d81dffb75b4a2cb0c4993b9092d4f68
検体 26	4b102b7043e76aa0a7772ed07564ef4e053a9e34533358e99ec4462c2839eb3a
検体 27	9609d597251570b3f0d0c839bd1166da1f9f202829aa7a98628cd74e1d97faee
検体 28	f38057389b2c4e84cd7723fa449e704910ccdfb20f244f53f28994b745bdaf50
検体 29	1013e2ef79f573ebfcbc02cde718c53e0f469508448b9f94592e5dc97bc8552f
検体 30	0593577650f2c8c2705adc41b3398caf04bf216e942836b560d2f3a79f39e9f6
検体 31	0fcb98b565b7b9281fdd05c0e0734f9bd0c64449975d231d5b90f621d3291952
検体 32	7b3f3ce769ad485244f45c5d3c1a2baa7f80c021c9efe35ade7b4e45e2433640

検体番号	SHA-256 ハッシュ値
検体 33	7ea7aefbac25f28471380b10f818b19b5a08894118f5e2ab58a323a3b40edf42
検体 34	8d5348efa4100eeda21b3df15cd2523ca01a9458c9da64788c0d289e0137e98f
検体 35	8d5348efa4100eeda21b3df15cd2523ca01a9458c9da64788c0d289e0137e98f
検体 36	8e1a0c2f344b1c9408932ce1c688193b80567adc3f42ed244a362734511699e3
検体 37	8f0d118672de43c64fada3cbc82cf27c20d1622d3e2c7dc96d5142133243fbc5
検体 38	9a7655c2156d590411f8236ef2468e62d7c210d96309c40f27a5343298e97391
検体 39	9a7655c2156d590411f8236ef2468e62d7c210d96309c40f27a5343298e97391
検体 40	aeb68012bda20b3d5b93903b72bb2745cd25117c42ffef395c06d2aa5a65957b
検体 41	aeb68012bda20b3d5b93903b72bb2745cd25117c42ffef395c06d2aa5a65957b
検体 42	aeb68012bda20b3d5b93903b72bb2745cd25117c42ffef395c06d2aa5a65957b
検体 43	b461ca50b63d345a0b10f580aa0bd8b8115468d8315844b53b91759b64b273dc
検体 44	d2d04181362a11e9f34802bec800aaf9f3a6210a0629793bc5b5cce71909c6d8
検体 45	d6a4675dd081236872b46109a1befc3ae61b122a74b2f0b2ea8c19ce062ea4c8
検体 46	de27ec703af00d1a5ccf27b5460c50e2c53e6bf614fd94fba30dfc7a9e6f3132
検体 47	de27ec703af00d1a5ccf27b5460c50e2c53e6bf614fd94fba30dfc7a9e6f3132
検体 48	ef72947e6cd9d74b8fe7c3164912f4594f8b53464908c4c33240b4a8f5150d61
検体 49	fa1b1d37b22e41c20bce779f891dfa108f85a3cc96d69e782c24dba326b145d8

図 目 次

表 目 次