IoT ネットワークにおけるネットワーク独立型異常検知および対策手法の研究

研究代表者　　山本　嶺　　電気通信大学大学院情報理工学研究科 准教授

1 はじめに

近年，無線通信技術や集積技術の発展により，通信機能を備えた様々な小型端末が普及しており，それらを用いた新しいネットワーク形態である IoT（Internet of Things）が急速に普及してきている．一般に IoT ネットワークにおける通信は，M2M（Machine to Machine）で行われることが多く，人による細かな制御を不要とした効率的な運用，管理が可能となっている．一方，一般に IoT ネットワークで用いられている機器の多くでは，センシングや機器への指示機能など必要最小限の機能のみを備えたものが多いことや，十分なリテラシーを有していないユーザによる利用も想定されることから，セキュリティ対策不足が懸念されている．実際に，初期設定のまま利用を続けたことにより，DDoS（Distributed Denial of Service）攻撃の踏み台として利用されるという攻撃が発生した．これは，図 1 に示すように Mirai[1]と呼ばれるマルウェアに感染した IoT 機器を利用して攻撃を行うものであり，機器のセキュリティ対策不足およびユーザのリテラシー不足に起因して発生したものである．これに対し，従来より IDS（Intrusion Detection System）などを用いてネットワーク内に進入した脅威を受動的に検知する仕組みが提案され，実システム等でも積極的に導入が行われている．また，IoT 機器自体に脆弱性がある場合には，ベンダーなどからファームウェア更新等によって対応することが一般的に考えられる．一方，これらを正しく運用し，脅威を未然に防ぐためには，IoT 機器を利用するユーザが十分なリテラシーを有することが必要であり，一般ユーザへの IoT 機器の浸透が想定される昨今の情勢では，IoT 機器の普及に伴って脅威も拡大するこが想定される．

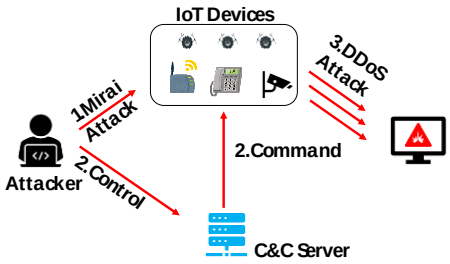

図 1 Mirai による攻撃例

本研究では，このような背景の下，一般ユーザにその存在を意識させることなく機器の異常検知を行い，異常発生時には自動的に不具合を修正する仕組みを提案する．そのため，本研究ではまず異常動作や異常トラヒックの検知を外部的にを行う仕組みについて検討を行い，検知した異常の原因を特定，解消するための方式検討を実施する．

2 研究背景と関連技術

2-1 研究背景

今日では，Internet of Things（IoT）機器が世界中で幅広く用いられている．IoT はあらゆるもののインターネットや産業用インターネット（IIoT）とも呼ばれる新しい技術であり，様々な機械や装置といった”モノ”がインターネットに接続し，互いに相互作用することができる技術である．また，IoT 技術は将来の技術分野において重要な技術であり，幅広い産業から注目を集めている．

IoT 技術を用いた機器は，家や企業，大学，近未来都市などの場面で用いられ，スマート環境を作り出すと考えられている．例えば，スマートな家を目指したスマートハウスという言葉がある．スマートハウスとは，IoT 技術を家に適用することで，家庭内のエネルギー消費の最適化やより快適さを求めた住宅を意味する．具体的には，IoT 機器がセンサによって家庭内の温度や湿度などを管理し，必要に応じて自動的に調節することで，快適に過ごす環境を提供することができたり，電気やガスなど家庭内のエネルギー消費を最適に制御することで省エネの効果を生み出す．

企業においても IoT 技術が活用されており，企業が保有する工場に IoT を活用することでスマート工場を実現する．スマート工場とは，生産性の向上や品質管理の向上を図ることを指す．このスマート工場を構築するためには，ビッグデータ解析技術や AI，IoT の導入が必要不可欠であり，IoT やビッグデータによって

公益財団法人電気通信普及財団
研究調査助成報告書　第36号　2021年度

品質・状態などの情報を基に工場内の設備同士，設備と工場内の作業員並びに管理者が連携することで高収益モデル並びに効率的な生産性を実現させることができる．

IoT が幅広い分野で急激に普及していく中で，新たなセキュリティ問題が発生していることも事実である．文献[2] では，IoT 機器には十分なセキュリティ対策がなされておらず，不注意なプログラム設計によってマルウェアやバックドアがインストールされやすいセキュリティ脆弱性が発生していると述べている．また，IoT 機器の数の急増によって IoT を扱う分野の規模が非常に大きくなってしまい，このようなセキュリティ問題が今までよりますます複雑化，顕在化している．

例として，スマートハウスが浸透していくとますます多くの家庭が IoT 機器を備え付けるようになる．スマートハウスを目指した IoT 機器は一般の人たちが利用することを想定して作られているため，工場出荷時には覚えやすい簡易な ID やパスワードが設定されていることが多い．また，実際にこういった IoT 機器を利用する利用者の多くが十分な情報リテラシーをもっていないため，ソフトウェアの更新がおろそかであったり簡易で推測されやすい ID やパスワードをそのまま変更することなく利用しているケースも多い．こういった欠点を狙った攻撃がますます増加している傾向にある．

これらの通信技術の普及に伴って，これまで接続されていなかった自動車やカメラなどの機器も Wi-Fi や携帯電話網などを介しインターネットに接続されることで新たな脅威が

発生し，それに対するセキュリティ対策も必要となっている．経済産業省が平成 28 年に策定した IoT セキュリティガイドライン Ver.1.0 [3] によると，2015 年に開催された BlackHat USA でインターネットから自動車を遠隔操作させることが可能とする脆弱性が紹介された．ここでは，自動車のマルチメディアシステムのコントローラへインターネット経由で接続し，別のコントローラのファームウェアを書き換え，車載ネットワークプロトコルである CAN バス上で不正なコマンドを送信することで自動車のハンドルやエンジン等の遠隔操作に成功した．また，2012 年に開催された Breakpoint Security Conference では，ペースメーカ及び植込み型除細動器へのハッキングのデモが紹介された．ここでは，植込み型除細動器のワイヤレストランスミッタの脆弱性を利用し，近距離から植込み型除細動器に不正な動作を行わせることに成功した．このように IoT 機器を狙った攻撃により，個人情報が漏えいしてしまったりシステム障害が発生してしまう問題の他，人命に関わる問題も発生する恐れがある．

2-2 マルウェア Mirai

IoT の需要が急激に高まり，IoT 機器の数も急激に増加する傾向にある中で，IoT 機器はサイバー攻撃を増加させる強力な踏み台にもなり得る．近年，IoT 機器が関わるセキュリティ問題が多発していることで IoT 機器には多くの脆弱性が潜在しているといえる．また，IoT 機器は継続的にインターネットに接続されていることにより，常に攻撃の対象となっている．このように，脆弱性の多い IoT 機器が急激に増加し幅広い分野で使用されることで，容易に攻撃の踏み台などになり得ることから，Distributed denial-of-service（DDoS）攻撃の被害など多くのサイバー攻撃を引き起こしてしまう．

文献[1]では，Mirai ボットネットが様々な産業により安全な IoT 機器をと警鐘を鳴らしている．Mirai は，主にウェブカメラやドライブレコーダといった IoT 機器によって拡散する．感染を受けた IoT 機器は，その後他の IoT 機器へ辞書攻撃などを用いてログインと感染を試みる．その後，Mirai は脆弱性をもつ IoT 機器から対象となるサーバに対して DDoS 攻撃をする．Mirai ボットネットは，四つの主要な要素から構成されている．まず一つ目はボットである．ボットとは，機器を感染させるマルウェアであり，その目的は脆弱性をもつ IoT 機器への感染を拡大させることとボットを操作する人物またはボットマスタからの指示コマンドを受け取り攻撃対象となるサーバへ攻撃を仕掛けることである．二つ目はコマンド&コントロール（C&C）サーバである．C&C サーバとは，ボットマスタと呼ばれるボットを管理しておりこのボットマスタがボットの状態を確認したり新たな DDoS 攻撃を行う．三つ目はローダである．ローダは，新たなボット対象となる IoT 機器に直接通信を試みることでより多くの IoT 機器を感染させる役割をもつ．最後にレポート用サーバである．これは，ボットネットにある全ての IoT 機器に関する詳細をまとめたデータベースを管理しており，新たに感染した IoT 機器はこのサーバと直接通信を行う．

ここで，マルウェア Mirai によるボットネットの操作と通信について説明をする．マルウェア Mirai は，TCP ポート 23 番や 2323 番を通じてランダムにパブリック IP アドレスをスキャンする．その際，US ポスタルサービス，国防省などの IP アドレスはスキャン対象から除外される．図 2 は，Mirai ボットネットによる操作と通信を示しており，七つのステップにより攻撃が行われる．ステップ 1 は，Mirai に備わって

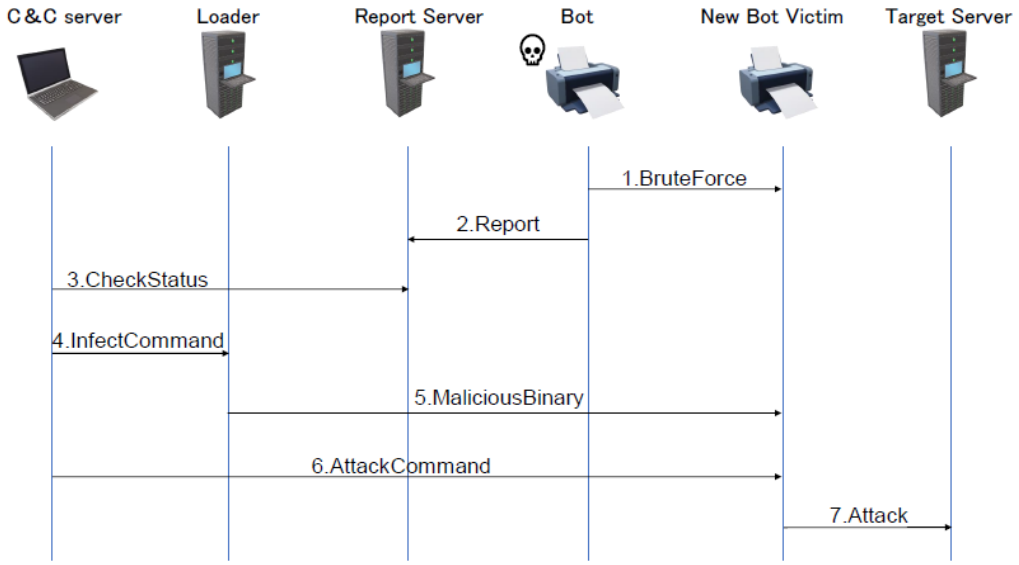

図 2 Mirai ボットネットによる操作と通信

いる 62 組のユーザネームとパスワードの組を利用し，デフォルトの資格情報をもつ脆弱な IoT 機器を発見するためにこれらの組み合わせを用いた総当たり攻撃を行う．ステップ 2 では，ログインに成功し，コマンドラインを取得した後,ボットはレポートサーバへログイン成功した IoT 機器に関する様々な特徴を送る.ステップ 3 では，C&C サーバを経由してボットマスタがレポートサーバと通信し，ボットネットの現在の状態を確認するとともに新たに侵入が可能な IoT 機器の探索を行う．ステップ 4 では，ボットマスタがどの脆弱性をもった IoT 機器を感染させるか決めた後，IP アドレスやハードウェアアーキテクチャなど全ての必要な詳細を含んだローダへ感染指示を送る．ステップ 5 では，ローダが対象となる IoT 機器へログインしマルウェア Mirai のバイナリバージョンのダウンロード並びに実行を行う．マルウェア Mirai が実行されるとマルウェア Mirai は他のマルウェアに感染させられないように Telnet や SSH サービスなどの侵入経路を断ち切ることでマルウェア自身を保護する．ステップ 6 では，ボットマスタが IoT 機器の種類や攻撃間隔，ボットならびに攻撃対象となる端末の IP アドレスといったパラメータに対応したコマンドを C&C サーバを通して送り攻撃対象の端末へ攻撃を行う．最後のステップとして，攻撃コマンドを受け取ったボットが TCP フラッド攻撃や HTTP フラッド攻撃など 10 種類の攻撃から一つ攻撃を攻撃対象の端末へ行う．これら一連のステップを踏んで攻撃対象の端末へと攻撃が行われる．

マルウェア Mirai の特徴的な部分は検知を回避する工夫がなされていないことである．全てのステップにマルウェア Mirai の痕跡が残される．例として，特定のポートに対して特定の資格情報を試していること，特定のバイナリコードをダウンロードしていることなどが挙げられる．

2-3 一般的なネットワークと IoT ネットワーク

一般的な通常ネットワークと IoT ネットワークは，それぞれ異なった特性をもつ部分もある．Non-IoT 機器と IoT 機器のトラヒックの特性の違いを理解することで，IoT 機器が他のアプリケーションに与える負担を軽減することや企業の IoT 分野の管理者が安全なアプリケーションに関する信頼性や遅延などの性能を適切に引き出すことができる．IoT 機器のトラヒックの特性を理解する最も重要な理由は，IoT 機器を踏み台とした大規模攻撃が多く行われており，IoT 機器のトラヒック特性を理解することで攻撃を特定するための手法を考えるのに有用であり，IoT 機器を対象とするサイバ攻撃を発見し抑制することが可能となると述べている．

一般に，Non-IoT 機器と IoT 機器では，使用するポート番号やポート数が異なる．図 3 に，Non-IoT 機器と 7 種類の IoT 機器が通信する際に使用するポート番号およびにポート数を示す．ポート番号の文字の大きさは使用頻度に比例している．例えば，部屋内の空気の品質を計測する Awair air monitor では，通信をする際に MQTT over SSL として登録されているポート番号 8883 を頻繁に使用し，反対に UDP プロトコルを用いたポート番号 1900 を使用する頻度は低いことを意味している．また，それぞれの機器名の右側に書かれている値は，その機器が通信をする際に使用するポートの数を示している．

例えば，Amazon が開発した人工知能搭載のスマートスピーカ Amazon Echo では，通信する際に 10 種類のポートを使用して通信を行っており，Non-IoT 機器では通信する際に 2382 種類のポートを使用している．この図から，Non-IoT 機器と IoT 機器の違いは，Non-IoT 機器では頻繁にポート番号 80 や 443 を用いて頻繁に HTTP 通信や HTTPS 通信をしているが，IoT 機器では HTTP・HTTPS 通信頻度は低いまたは使用していないことが分かる．また，Non-IoT 機器は非常に多くのポートを使用して通信を行っているが IoT 機器は少ないポート数で通信を行っている．これらの理由としては，それぞれの IoT 機器はあらかじめ決められた機能のみを搭載しているため，必要最小限のポートのみを用いると考えられる．

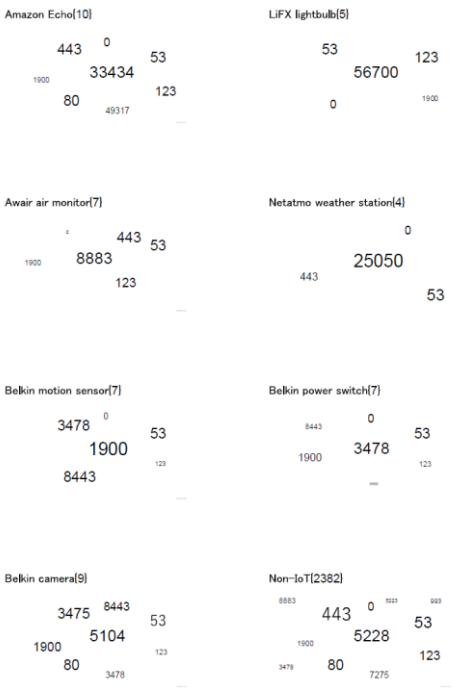

図 3 通信時に使用されるポート番号並びにポート数

2-3 一般的な脅威の検知および対策

一般に，ネットワーク内に存在する前述のような異常を検知するためには，シグニチャ型 IDS などを用いて，発生する特定の不正パターンを検知することが行われている．一方，これらはあらかじめパターンを登録しておく必要があることから未知の脅威/異常や動的に変化する脅威に対しては有効に動作しない場合がある．そこで，近年では機械学習を利用することでより的確に異常な動作を検知する仕組みが考えられている．機械学習は，主に入力データの与え方によって「教師あり学習」，「教師なし学習」，「強化学習」の 3 種類に大別される．また，機械学習のアルゴリズムとしては「線形回帰」，「ロジスティック回帰」，「SVM」，「ニューラルネットワーク」，「k-means 法」，「ランダムフォレスト」などが提案されている．異常検知を行う場合には，発生トラヒックや IoT 機器の挙動を入力データとして正常時の動作入力することで，それから外れた挙動を異常と判別したり，シグニチャ型と同様に異常時のデータを入力データとすることで異常を直接検知する方式などが考案されている．

検知した攻撃に対して対策を行う手法として，攻撃の種類に応じて自動的に対策を展開する方式が提案されている．文献[4]では，SDN (Software Defined Networking) を用いて，連結された Cloud と IoT ネットワークでグローバルなセキュリティポリシーを強化するためのアプローチが提案されている．これらのアプローチでは，network functionvirtualisation (NFV) と service function chaining (SFC) を利用することにより，セキュリティポリシーを実施する．想定されるユースケースとしては，ユーザが IoT デバイスに安全にアクセスする場合と，IoT インフラストラクチャの管理者がリモートクラウドやデータ分析サービスに安全にアクセスする場合が想定される．課題としては，セキュリティのアーキテクチャがカバーしている範囲は，IoT ネットワークの境界ルータまでであるこの状態では，IoT デバイスが安全であるということはできない．解決策としては，境界ルータから IoT デバイスの基地の脆弱性を監視し，侵害された IoT デバイスを残りのデバイスから隔離する手法が考えられる．しかし，これは Open Problem である．著者らは，NFV や SFC の処理を Virtual Machine (VM) からコンテナにすることによる軽量化を課題としていた．

3 ネットワーク独立型異常検知および対策

3-1 機械学習による異常検知

(1) 想定環境

本研究では，容易な検知機構導入を目的とし，前述のようにネットワーク独立型の異常検知を行うため，ネットワーク外部から観測可能な情報を基に機械学習を利用した異常検知方式の検討を行う．本研究で想定する環境は，図に示すように IoT 機器が属するネットワーク外部に監視機器を設置し，無線通信の漏れ聞き

(Overhearing)によって取得可能なデータを利用した異常検知を行う．

本研究では，機械学習による検知を行う前の検討として，従来より検討が行われている統計的な分析によって異常発生時に通信トラヒックの特徴分析を行った．ここでは，シドニー大学にて実施されて IoT ネットワーク攻撃監視プロジェクト[5]によって得られた通信データを用いた．この通信データには，10 種類の IoT 機器による通信が記録されており，7 種類の攻撃が記録されている．

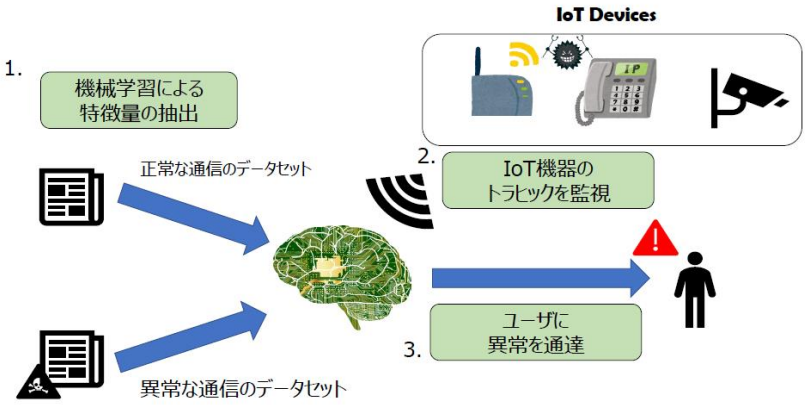

図 4 提案手法の概略

(2) 統計的分析

本研究では，外部から観測可能な通信特徴として，パケット送信間隔，パケット送信時間（パケットサイズに比例），単位時間当たりの最大パケット数，単位時間当たりのパケット数を利用して統計分析を行った．その結果，パケット送信時間において異常発生時にパケットサイズの大きいデータが送信されていると想定される特徴が検出された．一方，他の項目においては，有意な差が検出されず，正常時と異常時を明確に区別することができなかった．

(3) 機械学習による異常検知

次に，前述の 4 項目に対して機械学習による異常検知の検討を実施した．本研究では，機械学習アルゴリズムとしてサポートベクターマシン（SVM）を利用した学習を行い，入力データとして正常データをラベル 0，異常データをラベル 1 としたものを用いた．また，今回は基礎検討として 1 日分のデータを学習データとして利用し，テストデータも同様に 1 日分のデータを用いた．評価指標としては，混同行列を用いて TN，FP，FN，TP を評価し，正しく異常を検知できているかの評価を行った．

実験結果を図 5 に示す．結果より，今回用いたデータセットを利用した場合には，正常時と異常時を正確に分類することができていることが分かる．しかし，今回用いたデータセット数が少なく，正しく分類できていると考えることができるため，追加で検討を実施した．

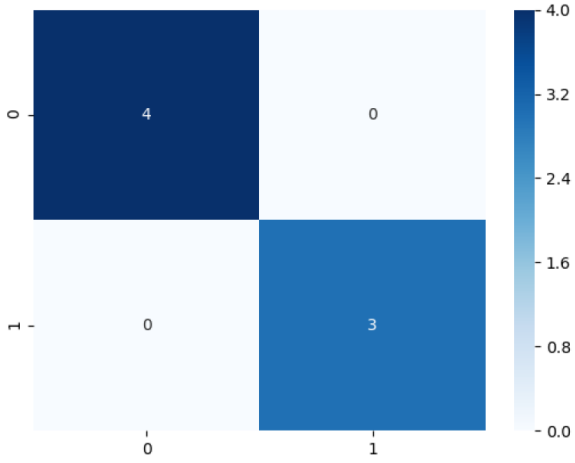

図 5 混同行列のヒートマップ

追加検討では，独立型の異常検知とは別にネットワーク内部に検知機構がある場合を想定し，5-tuple のデータを用いた場合と 5-tuple を一定の時間感覚で集計したデータを用いて異常トラヒックを正常に検知できるかの検証を行った．ここでも前述の方式と同様に SVM を用いて機械学習を実施した．その結果，異常トラヒックの識別精度としては 99%以上の精度で検知できることが確認された．

以上より，本研究が目的の一つとしていたネットワーク独立型異常検知は技術的に可能であることを示したとともに，ネットワーク内部に検知機構を導入することでも高い精度で検知が可能であることが分かった．

3-2 異常動作に対する対策

(1) IoT 機器の状態定義

IoT 機器が属するネットワーク正常に動作しているかどうかの判別については，前節の機械学習を用いた

検知によって判別可能であるが，IoT 機器自体が想定した動作をしているかまでは通信監視で判別をすることが困難である．そこで本研究では，Mozilla 社が提唱している WoT（Web of Things）という枠組みで利用されている Things Description を利用して IoT 機器の機能を定義する．ここで，Things Description とは，その IoT 機器が提供する Properties，Actions，Events を定義する．これにより，IoT 機器が本来想定している動作をしているかを外部から監視することが可能となる．本研究では，この Things Description で定義された各機器の動作と前述の異常検知の仕組みを組み合わせ，異常発生時に正常動作に回復させるための方式検討を行う．

(2) 実装検討

本研究では前述の回復動作を実現するため，IoT 機器を模した Raspberry Pi を用いて実装を行った．図 6 に初期実装の概要を示す．まず，外部機器から IoT 機器に対して定期的に監視を行うため，Prometheus を参考にした監視ソフトウェアを実装した．監視ソフトウェアでは，追加された IoT 機器から Things Description を取得し，そこに記述されているサービスのパスを監視対象として追加する．例として，照明機器の Things Description では properties として照明機器の状態や明るさの値を提供する．監視ソフトウェアは，毎秒ごとなど定期的に IoT 機器にアクセスを行い，サービスが正常に稼働しているか確認を行う．

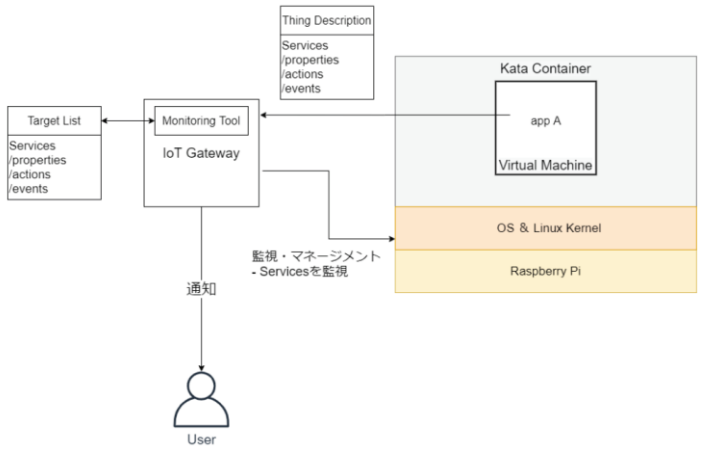

図 6 初期実装概略図

また，IoT 機器側でも提案動作を実現することやセキュアな端末環境を提供するため，IoT 機器が提供する各サービスをコンテナアプリケーションで動作させるように実装を行った．初期実装では，ホストとカーネルを共有しないセキュアなコンテナランタイムである Kata Containers[6]を利用した．Kata Containers は，軽量な仮想化技術を用いることでネットワーク，I/O，メモリなどの分離を行うことが可能であり，ここでは仮想化技術として Firecracker[7]を使用した．

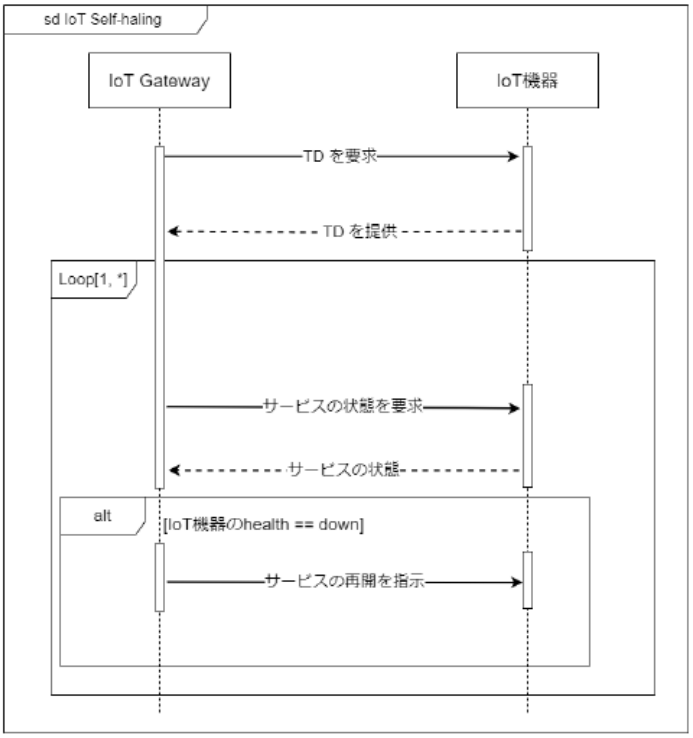

図 6 初期実装概略図

図 7 に IoT 機器監視とコンテナ再起動のシーケンス図を示す．初期実装では，外部装置は Things Description に基づいて IoT 機器の状態監視を行い，正常に動作していないと判断した場合，初期状態へと回復させる動作を行う．初期実装では，外部から IoT 機器内のコンテナを操作するため，HTTP を介してアクセス可能な RESTful API である Docker Engine API を使用した．初期実装により，異常が発生した機器のリセットが可能になったが，この段階では再度同様の異常が発生することが考えられる．そこで，異常に陥った原因を特定し，根本的な改善を行うための方式検討を行った．

(3) Audit Log を利用した因果関係分析

前述のように，単に IoT 機器をリセットを行うだけでは異常の原因となる要素が再び生じる可能性がある．そこで次の検討として，リセットを行う前に原因調査を行う必要がある．また，サイバー攻撃を受けた際にどの程度影響範囲があるかを調査する必要もある．サイバー攻撃の根本的な原因を調査する手法として,監査ログを用いた因果関係分析がある．因果関係分析では，監査ログ同士の因果関係を分析し，因果関係グラフとして出力する．監査ログを用いた因果関係分析のアルゴリズムを図 7 に示す．因果関係分析を行う前に，監査ログに detection point を設定する．detection point は，人がインシデントを発見した際のイベントをログに記録されているイベントに結びつける点になる．detection point を起点に，因果関係の調査を行う．次に，監査ログをイベントとして読みこむ．次に，グラフに含まれるオブジェクトを取り出す．E がオブジェクト O の時間しきい値によって O に影響を与えており，E の source オブジェクトがグラフに存在しない場合，E の source オブジェクトをグラフに追加する．E の source オブジェクトに E の時間を閾値として設定する．E の source オブジェクトがグラフに存在する場合は，E の source オブジェクトから E の sink オブジェクトへのエッジを追加する．本研究では,，因果関係分析の代表的な手法である BackTracker[8]を用いる．監査ログを記録するシステムとしては，LinuxAudit システムを使用する．Linux Audit システムは，カーネルで実行されたシステムコールをシステム管理者が設定したルールにマッチングしたものを，監査ログとして保存する．本研究における実装では，Yocto Project を用いて，balenaOS に Linux Audit システムのインストールを行った． Auditd はユーザランドで監視を行うルールの設定を行う．ユーザによって設定された監視のルールを用いて，カーネルのシステムコールの中でルールにマッチするものを audit log として記録する．

```
foreach event E in log { /* read events from latest to earliest */
    foreach object O in graph {
        if (E affects O by the time threshold for object O) {
            if (E's source object not already in graph) {
                add E's source object to graph
                set time threshold for E's source object to time of E
            }
            add edge from E's source object to E's sink object
        }
    }
}
```

図 7 因果関係分析

また，本実装ではよりセキュアな実行環境を提供するため，TEE（Trusted Execution Environment）による実装を行った．TEE は，ハードウェアによって提供される隔離実行環境であり，これを実現するためには CPU 自体が OS から隔離した環境を用意する必要がある．本実装では，TEE の一種である OP-TEE[9]を用いた．OP-TEE OS のユーザランドでは，TAs（Trusted Applications）と呼ばれるアプリケーションが動作する．TAs には，二種類の TA が存在する．一つは Secure World で信頼されたアプリケーションとして実行される user mode TAs である．user mode TAs は TEE Internal API を通じて，OP-TEE のカーネルと通信を行う．もう一つは，OP-TEE Core によって Secure World 外に公開されるインターフェースの Pseudo TAs である．Pseudo TAs は OP-TEE Core と同等の特権レベルを持ち，特定の目的のために使われる．Normal World で動作する Client Application（CA） は TEE Client API を利用して，Secure World で動作する TA にアクセスを行う．Cortex-A プロセッサでは，セキュアモニタとよばれるソフトウェアによって，Normal World と Secure World の切り替えを行う．更に，ログ解析のために SPADE[10]を利用した．

本実装では，Normal World で実行されたシステムのイベントをログとして記録するため，Linux Audit を利用した．Audit はシステムで実行されたシステムコールやファイルへのアクセスをイベントログとして生成し，イベントログを/var/log/audit/audit.log ファイルに書き出す．生成されたイベントログを安全に管理するため，Audit が生成したイベントログを TA と共有し，TA が遠隔のサーバに送信する機能を追加した．Audit によって生成されたイベントログを Normal World から収集し，遠隔のサーバに送信する TA を TA-Collect と呼ぶ．カーネルから TA-Collect の呼び出しには，OP-TEE が用意している API を利用した．

図 8 に既存のログの取得の流れと提案手法でのログの取得方法を示す．また，図中の色の濃さはセキュリティ強度を示す．Audit の具体的な処理の流れを説明する．まず，アプリケーションからシステムコールが呼び出されると，カーネル内の audit filter があらかじめ定義されたルール(audit rule) にシステムコールがマッチするかを判断する．システムコールがルールにマッチしていた場合,イベントログが生成される．イベントログには，Audit の記録のタイプ，タイムスタンプ，実行されたシステムコール，pid などの情報が含まれる．生成されたイベントログは buffer に追加される．カーネルスレッドとして起動する kauditd が buffer に格納されたイベントログを netlink を通じて，auditd に送信する．イベントログを受け取った auditd はイベントログをファイルに書き込みを行う．

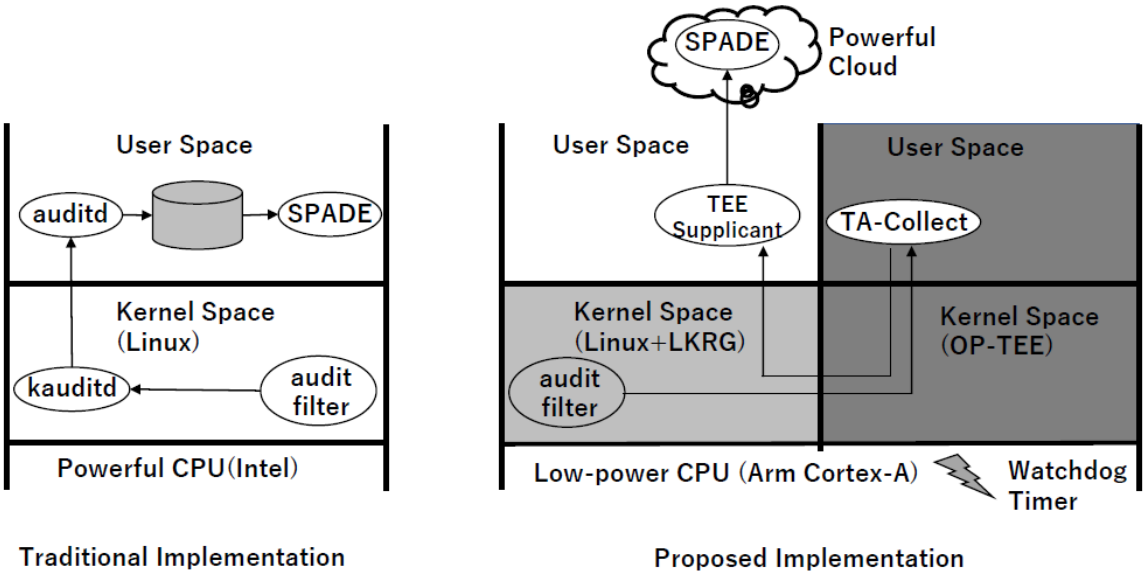

図 8 従来実装と本実装の比較

audit_log_end によって audit filter にマッチしたシステムコールがイベントログとして生成されるたびに TACollect が呼び出される．仮に，カーネルと TA-Collect の間にセッションが存在しない場合，tee_client_open_context と tee_client_open_session 関数を利用し，TACollect とのセッションを作成する．次に Secure World と Normal World の両方からアクセスできる共有メモリを tee_shm_alloc 関数を用いることにより作成し，イベントログを共有メモリへとコピーする．最後に，カーネルから tee_client_invoke_func 関数を用い，TA-Collect の呼び出しを行う．カーネルから呼び出された TA-Collect は共有メモリのイベントログを OP-TEE OS のメモリーへとコピーする．次に，TCP を利用し，コピーしたイベントログを遠隔のサーバへ送信する．TA-Collect から遠隔のサーバへの接続は TLS（Transport Layer Security）を使用する．

提案手法のログを解析する環境として，SPADE を利用した．SPADE はクロスプラットフォームに対応し，収集された分散データの出所の追跡や分析する機能を提供するシステムである．分散データの収集には，SPADE が提供する各プラットフォームに対応したレポータが用いられる．レポータはシステムコールの監視を行い，それらを実績セマンティックスに変換する．SPADE のサーバはこれらのレポータから送られてくる実績セマンティックスの統合を行う．各レポータは SPADE サーバの一部として実行されており，SPADE が動作するホスト以外からのログをリアルタイムに処理するレポータが用意されていない．そのため，別のホストから収集した監査ログをリアルタイムに受け取るための新しいレポータを作成した．具体的には，Audit レポータでは，/var/run/audispd_events に UNIX ドメインソケットで接続を行い，ログの取得を行っている．TA から TCP を経由してサーバに送られてきたログは，/var/run/iot_events にポートをオープンしているサーバに書き込みがされる．新しく作成したレポータはこのポートに UNIX ドメインソケットで接続を行い，ログの収集を行う．ログの解析として，SPADE のクエリエンジンを利用して攻撃者の行動を追跡する．SPADE のクエリエンジンでは，保存された実績セマンティクスに対してクエリを発行し，クエリとマッチする値を持つエッジを取得することができる．さらに，指定したグラフ内のエッジと関連があるエッジを探索することもできる．クエリから取得したグラフは，Graphviz[] を利用してグラフの可視化を行った．また，SPADE のレポータやクエリエンジンは Intel x86 アーキテクチャの 64bit にしか対応していないため，ARM アーキテクチャの 64bit をサポートする機能を追加した．

以上の実装により，IoT 機器に異常が発生した際に単にリセットを行うだけでなく，異常の原因となった箇所を特定し，対策を行うことを可能にした．一方，今回の実装ではセキュアな実行環境の提供と要因分析を実行可能な環境の提供に留まっているため，攻撃により IoT 機器に異常が生じた際に回復を行う箇所の検討が必要となる．しかし，攻撃を受ける IoT 機器は多岐に渡ると考えられるため，全ての機器に適用可能な統一的な対策は困難であると考えられる．

4 むすび

本研究では，近年急速に普及が進んでいる IoT 機器に対する脅威に対処するため，異常動作の検知および異常から回復させるための対策に関する検討を実施した．

まず本研究では，異常動作を検知するため，統計的な分析，ネットワーク独立型異常検知の検討を行い，外部から観測可能な指標によって攻撃などで異常が生じた場合の状態判別を可能にする手法を提案した．一方，外部からの状態判別では，ネットワーク全体の健全度を測ることは可能であるが，ネットワークが正常に動作していると判別された場合においても個々の IoT 機器に着目した際には正しく動作していない可能性が考えられる．

そこで，本研究では WoT（Web of Things）で利用されている Things Description と呼ばれる各 IoT 機器ごとに割り当てられる動作に関する情報を利用し，外部から Things Description と IoT 機器の動作を照合することで各 IoT 機器が正常に作動しているか検証する仕組みの考案を行った．これにより，Things Description と実際の動作が乖離している場合には異常が発生している機器を特定することを可能にし，該当 IoT 機器のリセットによって動作回復を行う方式の実装を行った．

更に，IoT 機器のリセットのみでは再び同様の異常が発生することが想定されるため，異常原因の特定を可能にするため因果関係分析を行う仕組みとよりセキュアな実行環境を提供する TEE を利用した実装によって，IoT 機器自体を堅牢化するとともに異常原因追及を可能にする枠組みを提案した．

今後は，本研究で実施した成果を基に，外部からの異常検知をより高解像度で実行可能な方式の検討を行い，異常原因特定に役立てることや，特定された異常原因を排除するための効果的な方式，端末環境の差異を吸収可能な実装方式について検討を行う．

【参考文献】

[1] C. Kolias, G. Kambourakis, A. Stavrou, and J. Voas, "DDoS in the IoT:Mirai and Other Botnetss," IEEE Journal of Computer,vol.50, no.7, pp.80-84, Jan. 2017.

[2] Z.-K. Zhang, M.C.Y. Cho, C.-W. Wang, C.-W. Hsu, C.-K. Chen,and S. Shieh, "IoT security: Ongoing challenges and research opportunities," IEEE 7th International Conference on Service-Oriented Computing and Applications, Nov. 2014.

[3] 総務省, "IoT セキュリティガイドライン ver 1.0," https://www.meti.go.jp/press/2016/07/20160705002/20160705002-1.pdf, July 2016.

[4] R. Duan, A. Bijlani, Y.Ji, O. Alrawi, Y. Xiong, M. Ike, B. Saltaformaggio, and W. Lee, "Automating patching of vulnerable open-source software versions in application binaries," Network and Distributes Systems Security Symposium 2019, pp.1-15 San Diego, CA, USA Feb. 2019.

[5] A. Sivanathan , H.H. Gharakheili , V. Sivaraman, and A. Hamza, "IoT Security," https://iotanalytics.unsw.edu.au/

[6] About kata containers — kata containers. https://katacontainers.io/.

[7] Firecracker. https://firecracker-microvm.github.io/.

[8] S. T. King and P. M. Chen. "Backtracking intrusions," In Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 223–236, 2003

[9] Op-tee. https://github.com/OP-TEE.

[10] Ashish Gehani and Dawood Tariq. "SPADE: Support for Provenance Auditing in Distributed Environments," In ACM/IFIP/USENIX International Conference on DistributedSystems Platforms and Open Distributed Processing, pp.101–120, 2012.

[11] Graphviz - graph visualization software. http://www.graphviz.org/.

〈発　表　資　料〉

題　　名	掲載誌・学会名等	発表年月
IoT 機器の振る舞いに着目した異常動作からの自動復旧に関する検討	WIDE プロジェクト研究会	2020 年 12 月