Vol.2022-OS-154 No.3
2022/3/14

ZT-IoT: ゼロトラストIoTのための システムソフトウェア構築に向けて

竹房 あつ子[1,4,a)]　五十嵐 淳[2]　関山 太朗[1,4]　松井 俊浩[3]　小野 泰司[3,1]　福田健介[1,4]
蓮尾 一郎[1,4]　合田 憲人[1,4]　石川 裕[1,4]

概要：Society 5.0 では，クラウド，エッジ，IoT デバイス群からなる IoT システムと AI 技術を活用して，様々な社会的課題の解決や新たな価値創造が期待されている．しかしながら，サイバー攻撃の高度化，多様化により，社会インフラのサービス停止といった甚大な被害が発生するようになり，IoT システムにおいても「ゼロトラスト」を前提としたシステムの設計，配備が急務となっている．本稿では，ゼロトラスト IoT (ZT-IoT) システムの実現に向け，特にシステムソフトウェアに関する研究課題を明らかにする．まず，IoT アプリケーション事例と前提とする IoT システムモデルを示し，IoT システムにおけるセキュリティ上の脅威や信頼性，レジリエンスを実現するための課題を述べる．そして，それらの課題を解決する ZT-IoT アーキテクチャを定義し，RoT，セキュアブート，TEE，監視，アクセス制御，トラストチェーン，ソフトウェアの信頼性保証とライフサイクル管理，および，形式手法によるレジリエンスとアカウンタビリティについて議論する．

1. はじめに

Society 5.0 時代では，多種大量のセンサから収集された膨大なデータに対して人工知能技術（AI）を活用し，エネルギー問題や超高齢化社会，地域格差等の様々な社会的課題を解決や，新たな価値の創造が期待されている．このような IoT（Internet of Things）システムは，広域のネットワークに接続された IoT デバイス群とクラウドまたは組織内の計算機で構成され，ファイアウォールや VPN（Virtual Private Network）などのネットワークの境界を適切に設定することでシステム全体の安全性を確保するのが一般的であった．しかしながら，サイバー攻撃の高度化，多様化し，ハードウェア，OS，ネットワーク，クラウドプラットフォームに内在する様々な脅威も顕在化している．2021 年 5 月には，米国石油パイプライン最大手の Colonial Pipeline が VPN への不正侵入を受けてすべての業務を 6 日間停止し，ガソリンが品薄になるといった甚大な被害が発生するなど，サイバー攻撃が社会インフラに大きな被害を与える事例が複数発生している．日本政府も，2022 年度から重要インフラ事業者に対してサイバー攻撃への備えを義務付ける方針を示している．よって，ネットワーク境界等の暗黙の信頼を前提としないセキュリティ対策が不可欠となっている [1]．

Google では，BeyondCorp[2] と呼ばれるエンタープライズセキュリティのためのアプローチを 2014 年に発表し，ネットワークの境界による安全性に頼るのではなく，個々のユーザやデバイスに対して認証・認可や暗号化通信などを用いて動的に信頼するアクセスモデルを採用している．また，2020 年 8 月には米国標準技術研究所（NIST）からゼロトラストアーキテクチャ（Zero Trust Architecture）と呼ばれる概念定義が発表され [3]，情報システムやサービスの構成要素間のすべての通信を保護し，セッション単位での認証認可と，継続的な監視によるセキュリティ対策の改善等を含むサイバーセキュリティ計画の方法論が示された．複数組織が協力し，NIST の定義するゼロトラストアーキテクチャのサンプル実装を開発する試みもある [4]．

しかしながら，特に IoT システムにおけるゼロトラストアーキテクチャの実現には以下のような課題がある．ゼロトラストの定義は大まかな指針にとどまっており，具体的実装の一般的方法は明らかでない．また，Google 等の既存のゼロトラスト実装は，クラウドサービスの利用を前

[1] 国立情報学研究所
National Institute of Informatics
[2] 京都大学
Kyoto University
[3] 情報セキュリティ大学院大学
Institute of Information Security
[4] 総合研究大学院大学
The Graduate University for Advanced Studies (SOKENDAI)
[a)] takefusa@nii.ac.jp

提としたものであり，IoT システムにそのまま適用することができない．すなわち，IoT システムでは人が介在する多要素認証や生体認証情報の利用はできない，パソコン等のユーザ端末と比較して性能面，電力面での制約も大きいといった課題がある．SIP/IoT 社会に対応したサイバー・フィジカル・セキュリティ [5] プログラムでは，IoT デバイスの製造から運用までのサプライチェーン全体を対象とした研究開発を進めている．FIDO[6] では，IoT デバイスを承認するプロトコルについて議論している [7]．しかし，いずれもシステムソフトウェアの観点で十分に議論されているわけではない．さらに，Society 5.0 の実現のためには IoT システムセキュリティを説明可能な形（アカウンタビリティ）で保証することで，社会受容を促進することが不可欠である．

本稿では，ゼロトラスト IoT システムの実現に向け，特にシステムソフトウェアにおける研究課題を明らかにする．まず，IoT アプリケーション事例の紹介と本稿で前提とする IoT システムアーキテクチャを定義する．次に，IoT システムのセキュリティ上の脅威と，信頼性，レジリエンスのための課題を明らかにする．その後，ゼロトラスト IoT のためのシステムアーキテクチャを示し，システムソフトウェアに関する各研究課題について議論する．

2. IoT アプリケーション事例とシステムアーキテクチャ

IoT アプリケーション事例を示すとともに，本稿で想定する IoT システムアーキテクチャを定義する．

2.1 IoT アプリケーション事例

IoT を活用した応用事例は多岐に渡る．Society5.0 では，IoT を含む様々データから創造される新たな価値の事例として，交通，医療・介護，ものづくり，農業，食品，防災，エネルギーをあげており，入力となる IoT データの種類と収集したデータの解析結果の出力となるサービス事例を紹介している [8]．また，それらを含む都市・地域全体のデジタル化を図ることを目指し「スマートシティ」の推進と SDGs 達成を提言している McKinsey Global Institute のレポート [9] では，Human，Home，Retail environments，Offices，Factories，Worksites，Vehicles，Cities，Outside で分類し，潜在的な経済的インパクトについて議論している．IoT のための 5G 技術に関するサーベイ論文 [10] では，スマートシティ，スマートホーム，インダストリ，ヘルスケア，ITS（Intelligent Transportation Systems）で分類している．IoT セキュリティに関するサーベイ論文 [11] ではスマートシティ，スマート環境，スマート測定・グリッド，セキュリティと緊急事態，スマートリテイル，スマート農業・畜産，ホームオートメーションと目的ごとに分類している．これらの論文 [10], [11] では，分類ごとに具体的な事例を挙げている．

本稿では，IoT データを収集する場所に着目してスマートホーム／工場，スマートシティ，スマート農業・畜産／環境の 3 つに分類し，国立情報学研究所が実施している SINET 広域データ収集基盤実証実験（モバイル SINET）[12] の参加プロジェクトを含む，いくつかの事例を紹介する．

2.1.1 スマートホーム／工場

スマートホームでは，防犯，エネルギー利用効率最適化，生活の質の向上を目的とした事例があげられ，すでに多くの商用サービスも提供されている．学術的な試みとして，お茶の水女子大学ではプライバシを考慮した室内見守りサービスを想定し，エッジ，クラウド分散動画像解析手法を提案している [13]．名古屋大学では，屋内環境における自律移動ロボット（AMR）を支援，管理，監視，検証するための自律移動ロボット用デジタルツインシステム DigiMobot を開発しており，DigiMobot と開発した 2 種類の自律移動ロボットを用いた倉庫内での実証実験を行っている [14]．

2.1.2 スマートシティ

スマートシティは，交通の最適化や都市環境の観測，防犯防災といった，都市生活環境の向上を目的とした事例がある．特に自動車は，運転ミスによる交通事故や環境問題，労働人口減少等の様々な社会問題を解決するために，コネクティットカーや V2X（Vehicle-to-Everything）無線通信技術の活用が期待されている [15]．ANL では，データプライバシを考慮して都市環境の監視，解析を可能にする，多種センサを搭載したエッジコンピューティング端末とそのためのソフトウェアプラットフォーム Waggle を開発している [16]．モバイル SINET を用いた研究では，大分大の新幹線の軌道回路の信頼性向上と保守のための線路の振動監視・解析 [17] や，新潟大の視覚障碍者の屋外歩行支援や災害発生時の位置情報提供を目的とした高精度測位実験 [18] がある．また，広島大では人の脳波データや関連する画像情報，位置情報を収集し，人の感情を推測して様々なサービスに活用する IoT システムの構築を目指している [19], [20]．

2.1.3 スマート農業・畜産／環境

スマート農業・畜産はビニールハウス内，田畑，牧場に IoT デバイスを設置し，農作物や畜産物の生産性や品質を高めたり，生育環境の最適化を目的としている．スマート環境は山海川といった広域環境に様々なセンサを設置し，観測情報を自然監視や災害予測，防災等に利用される．KDDI 総合研究所のグループでは，鳥獣による森林被害防止のため，防鹿柵に加速度センサを設置して LPWA（Low Power Wide Area）でゲートウェイ装置にセンサデータを集約し，クラウドに送信・解析・遠隔監視を可能にした [21]．また，東京大学では野生動物にウェアラブルデバイスを装

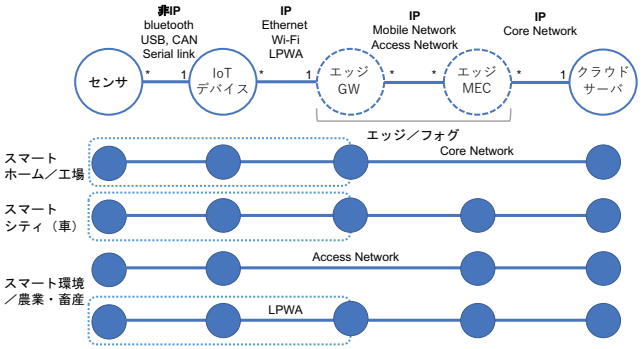

図 1　IoT デバイス，エッジ，クラウドで分類した横方向の IoT システム階層モデル．

着し，動物間でのデータ共有により環境モニタリングを長期的に行うシステムを開発している [22]．

2.2　IoT システムアーキテクチャ

IoT システム階層は様々な論文 [11], [23], [24], [25], [26] や標準仕様 [27] で定義されているが，システムソフトウェアの課題を明らかにする上で十分な階層定義がなされているとは言えない．本稿では，IoT デバイス，エッジ，クラウドの横方向のシステム階層と，センサ／物理，通信，オペレーティングシステム（OS），ミドルウェア，アプリケーションの縦方向のシステム階層の両方を定義する．

2.2.1　横方向のシステム階層

IoT システムは，IoT デバイス，エッジ，クラウドで構成される [25]．エッジコンピューティングは，クラウドとエンドユーザ／デバイスの間の「ネットワークのエッジ」で処理をするという意味で使われており，フォグコンピューティングとも呼ばれている [24]．サービス遅延やデータ通信量の削減やデータの集約，計算負荷の分散，プライバシの保護といった様々な目的でエッジコンピューティングが適用される．また，IoT システムにおけるエッジは，主に IoT デバイス群のゲートウェイの役割を果たすエッジサーバと，モバイルネットワークの基地局のリソースを活用するモバイルエッジコンピューティング（MEC）サーバからなる [25], [26]．

図 1 にセンサ，IoT デバイス，エッジ，クラウドで分類した IoT システム階層モデルを示す．センサと IoT デバイス間は非 IP，その他は IP で接続される．エッジは，応用事例によっては利用しない場合もあるため，破線で表している．図中の最上段では，センサと IoT デバイスなどの接続数の関係を各ノード間をつなぐ辺の両端に表しており，1 は 1 つ，* は複数あることを表す．すなわち，1 つのクラウドサービスに対して複数の MEC サーバが存在する，MEC とエッジ GW はそれぞれ複数存在しうる，エッジ GW と IoT デバイスおよび IoT デバイスとセンサの関係は 1 対多になっていることを表す．

図 1 の 2 段め以降では，各応用事例の一般的な実装で

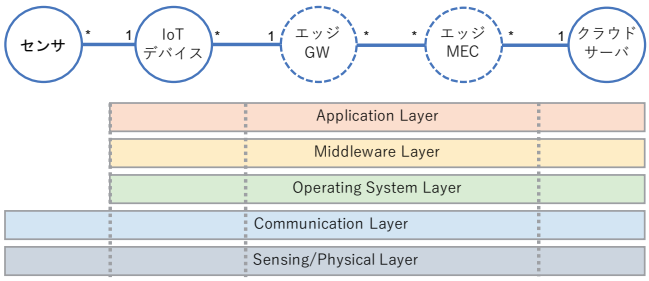

図 2　本稿で想定する IoT システム階層モデル．

はどのように構成されているかを示す．スマートホームおよびスマート工場では，家または工場内に複数の IoT デバイスが設置されており，エッジ GW で各種センサデータを集約してクラウドへデータを送信する．遅延時間の低減やプライバシや秘密保護の観点で，エッジ GW サーバで匿名化のようなデータの前処理や AI 処理を行う場合もある．スマートシティにおける交通サービスでは，車内に複数のセンサおよび IoT デバイスがあり，車内の情報をエッジ GW サーバで集約して MEC サーバまたはクラウドへ送信する．交通サービスでは地域ごとに低遅延で情報を共有する必要があるため，MEC サーバが利用される．スマート環境およびスマート農業・畜産では，エッジ GW を利用する場合としない場合が想定される．ビニールハウス内や船内などで複数センサデータを収集・集約して送信する場合は，スマートホーム／工場と同様の構成となる．放牧など広域環境で利用する場合は，各 IoT デバイスから直接 MEC やクラウドに送信する場合もある．さらに，モバイルネットワークが利用できず電源供給も十分に行えないような環境では，LPWA でプライベートなネットワークを構築し，モバイルネットワークが利用できる環境で広域に分散した IoT デバイスから送信されたセンサデータをエッジ GW で集約して送信する，一番下のような構成となる．

2.2.2　縦方向のシステム階層

IoT システムの縦方向のシステム階層の比較として，サーベイ論文 [23] では 8 つの論文の階層分類を比較し，最終的にアプリケーション層，通信層，認知層の 3 階層で分類している．ただし，これらの分類では 2.2.1 節で述べた横方向の階層分類と縦方向の分類が混在している，システムソフトウェアを議論する上で重要なオペレーティングシステム層が表現されていないといった問題がある．

本稿では，センサ／物理層，通信層，オペレーティングシステム（OS）層，ミドルウェア層，アプリケーション層の 5 つの階層で分類する．**図 2** に横方向と縦方向の階層を考慮した IoT システム階層モデルを示す．センサ／物理層は，温度湿度計や GPS 等の個々のセンサ，IoT デバイスやエッジサーバのハードウェア，およびネットワークの OSI 参照モデルの L1（物理層）を表す．通信層は，OSI 参照モデルにおける L2（データリンク層）から TCP，UDP

を含む L4（トランスポート層）までを表す．OS 層はオペレーティングシステム，ミドルウェア層は MQTT 等の IoT のためのミドルウェアを示す．また，アプリケーション層はサービスごとに配備されたセンサ，エッジ，クラウドのプログラムを表す．IoT デバイスからクラウドサーバまでは，上記の 5 階層，センサはセンサ／物理層と通信層のみとなる．

3. IoT システムのセキュリティ上の脅威

IoT システムにおけるセキュリティ上の脅威は，図 2 に示す IoT システムモデルのすべての階層で発生しうる．本節では，我々が想定する IoT システムモデルの各階層で発生するセキュリティ上の脅威について，文献 [23] と同様にセキュリティの三要素である Confidentiality（機密性），Integrity（完全性），Availability（可用性）の観点で整理する．

3.1 アプリケーション層

アプリケーション層はネットワーク経由で IoT デバイスに攻撃を与える経路となる．アプリケーションの改竄は，i) アプリケーション自体の脆弱性によりアプリケーション実行中に改竄される，ii) リモート保守機能を使ってアプリケーションプログラムが置換される，の 2 つがあり得る．i) は，システムでアプリケーションの挙動を監視しアプリケーションの挙動が異常動作しているということが検知し対応できれば攻撃の被害は最小限に留めることが出来る．しかし，必ずしも挙動異常を検出できるとは限らない．

ここでは，アプリケーションの改竄やアプリケーションレベルでのアクセス制御が適切に設定されていない場合の脅威について，IoT デバイス内でのアプリケーションを以下の 3 種類に分類し説明する．リモート保守のための telnet や ssh サービスは 3.3 節で述べる．

- データ送信系
  IoT デバイスで集められたデータを外部に送信するアプリケーション．例としてはセンサーネットワークで使われる IoT デバイス がある．
- 制御・サービス系
  IoT デバイスが外部から制御あるいはサービス要求を受け付けて処理するアプリケーション．例えば，ロボットのリモート制御や監視カメラの首振り制御が挙げられる．IoT デバイスが有する情報を提供する処理もここに含める．
- データ受信系
  外部 IoT デバイスで集められたデータを受信するアプリケーション．例えば，自動運転において信号機などの IoT デバイスからの情報受信が挙げられる．

3.1.1 機密性

データ送信系アプリケーションの改竄により，本来送信してはいけない外部にデータが送信される可能性がある．ネットワークアドレスによるアクセス制御で意図しない外部との通信を遮断することは可能である．しかし，ネットワークアドレスとして許されるサイトであっても意図しない相手にデータを送信する可能性がある．

制御・サービス系アプリケーションではアプリケーションの改竄がなくても適切なアクセス制御が設定されていないことにより，不正サイトからの要求に応答しデータが漏洩する可能性がある．制御・サービス系アプリケーションの脆弱性に関しては，2022 年 1 月に報告された Tesla のダッシュボードアプリケーション脆弱性により不正に Tesla API を呼び出すことが可能になったという報告がある [28], [29].

上記情報漏えいを防ぐために, 強制アクセス制御 (Mondatory Access Control) をアプリケーションレベルで導入する必要がある．アクセス制御機構はミドルウェア層で実現できるだろう．アプリケーションレベルでは適切なセキュリティポリシを定義する必要がある．

3.1.2 完全性

データ送信系アプリケーションの改竄により，改竄したデータ送信が可能となる．データ受信系アプリケーションの改竄により，受信したデータを改竄することが可能となる．

このようなデータ改竄はシステムソフトウェア側では改竄そのものを検知することは困難である．改竄されたアプリケーションが実行されているかを検知してデータ改竄を未然に防ぐしかない．アプリケーションが実行前に改竄されたかどうかは，実行ファイルが改竄されていないか検査することで未然に確認できる．一方，アプリケーション実行中にコードが埋め込まれるような場合，システムソフトウェアの監視プログラムがアプリケーション実行の振る舞いの変化を検出できるならばアプリケーションの改竄が検知できる．

3.1.3 可用性

アプリケーションを改竄しアプリケーションを停止させることができる．アプリケーションの停止方法には，アプリケーション実行を終了させる，無限ループさせるなどして処理不能にさせる，などがある．これらの脅威は，アプリケーションの挙動を監視し対応できるだろう．

制御・サービス系アプリケーションにおいてシステム停止コマンドが外部から実行できる場合，アプリケーションの改竄がなくても適切なアクセス制御が設定されていないことにより不正サイトからの要求に応答しシステムを停止することが可能である．アプリケーション単位でのセキュリティポリシの設定が必要である．

3.2 ミドルウェア層

IoT におけるミドルウェア層として TCP や UDP 上の通

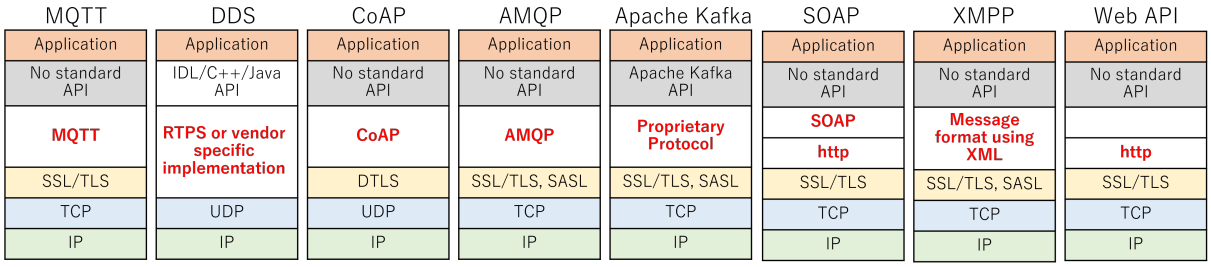

図 3 通信プロトコル

信プロトコルを考える．図 3 に示すとおり IoT で使われている通信プロトコルには，MQTT[30]，DDS[31]，CoAP[32]，AMQP[33]，Apache KAFKA[34]，[35]，[36]，SOAP[37]，XMPP[38]，http を使った Web API などがある．なお，SOAP も Web API の一つであるが他の Web API と違い標準化されているために区別した．MQTT，DDS，CoAP，AMQP，Apache Kafka はいずれも pub/sub 通信モデル [39] である．pub/sub 通信モデルでは，送信者 (Publisher) は受信者を特定せずメッセージを非同期に送信し，受信者 (Subscriber) は特定の送信者を想定せずにメッセージを受信できる．受信者が受信するメッセージの選択方法として topic-base がある．Publisher はメッセージを topic(キーワード (文字列)) に送信し，メッセージは topic 毎にグループ化される．Subscriber は topic を指定してメッセージを受信する． MQTT は IoT デバイスのセンサー情報を Cloud に送信するために広く使われている．MQTT では Publisher と Subscriber は Broker と呼ばれるサーバを介してメッセージ交換される．DDS は ROS2 ロボット OS で採用されておりロボット制御や自動運転に使われている．SOAP を使っている例としては，ONVIF (Open Network Video Interface Forum) が定義している監視カメラの首振り機能 (PTZ 機能) 制御プロトコルがある [40]．Web API は広く使われており，例えば，Tesla 社の車載情報機器は Web API で制御できる [41]．

このような通信プロトコルを実現しているライブラリが改竄されることにより以下の脅威が生じる．アプリケーション層における脅威と共通点も多い．

3.2.1 機密性

例えば，MQTT のように Broker が介在するような通信におけるデータ漏洩には，IoT デバイスが有するデータの漏洩と Broker に蓄積されているデータ漏洩がありうる．前者は IoT デバイスからデータが publish する時にデータが搾取される場合である．後者は IoT デバイスが任意の topic を subscribe してデータが搾取される場合である．後者の場合，Broker 側でアクセス制御機能があれば脅威を防ぐことが出来る．

このようなデータ漏洩は IoT デバイス側通信ライブラリの脆弱性だけでは成立しない．本脆弱性により搾取したデータを IoT デバイスの外部に流出させる脆弱性が必要となる．3.1 節で述べた通り，このような情報漏えいを防ぐためにアクセス制御機構が必要となる．MQTT のようなミドルウェア層の API を使ってデータが漏洩する場合にはミドルウェア層のアクセス制御機構で脅威を防げる．そうでない場合は通信層でアクセス制御することになる．これはミドルウェア層で通信プロトコルに依存しない統一した API を有するアクセス制御機構を実現すれば，アプリケーションに依存せずユーザはセキュリティポリシを決めることが出来るようになる．

3.1 節で述べた IoT デバイスにおける制御・サービス系アプリケーションは，HTTP プロトコルの PUT，GET，POST コマンドで実現される．ミドルウェア層において適切なアクセス制御を行うことでデータ漏洩を防ぐことが可能となる．

3.2.2 完全性

改竄された通信ライブラリにより，IoT デバイスが本来送信すべきデータとは異なるデータを送信することでデータが改竄される．アプリケーション層同様，ミドルウェア層でデータ改竄を検知するのは困難である．通信ライブラリが改竄されたかどうかを監視することによりデータ改竄の可能性を未然に防ぐしかない．

3.2.3 可用性

改竄された通信ライブラリは IoT デバイスからのデータ送信を止めることが可能である．改竄された通信ライブラリがデータを送信しない場合，アプリケーションレベルの挙動を監視していても本異常を検知できない．アプリケーションのデータ送信が定期的に行われているならば，定期的に通信が行われているか監視することにより異常を検知できる．

3.1 節で述べた制御・サービス系アプリケーションを実現している通信ライブラリが改竄され適切なアクセス制御が設定されていないと，不正サイトからの要求に応答しシステムが停止させられる．これはアプリケーション単位でのセキュリティポリシの設定により回避しなければいけない．

3.3 オペレーティングシステム（OS）層

IoT デバイス内では，3.1 節で述べた通りユーザアプリケーションに関係するプロセス以外にリモート保守のための telnet，ssh，OTA 機能を実現するプロセスが動作している．OS 層における脅威は機密性，可用性，完全性全般の問題となる．ここでは OS 層における攻撃経路の観点でまとめる．

3.3.1 不正侵入

リモート保守のために telnet や ssh 機能や Web サーバ機能が搭載されている IoT デバイスではデバイス設置時の安易な ID とパスワード設定により被害を被る．2016 年に発見されたサイバー攻撃 Mirai[42] は保守用に用意されていた telnet アカウントがデフォルト設定されたままの IoT デバイスに不正侵入し DDoS 攻撃に使われた．不正侵入された後，データ漏洩 (機密性問題)，データ改竄 (完全性問題)，DoS(可用性問題) の可能性が生じる．

Linux カーネルでは強制アクセス制御 (Mandatory Access Control) 機構として，LSM(Linux Sexurity Module) 上に SELinux, SMACK, Tomoyo, Apparmor, Yama が実装されている．アカウント毎にきめ細かい強制アクセス制御 (Mandatory Access Control) が設定されていれば，不正侵入された時に使用されたアカウントで許される操作で生じる脅威のみに限定でき，被害を最小限に抑えることが出来る．

3.3.2 不正システムコール使用

アプリケーションが改竄されて OS システムコールが呼ばれた時，アクセス制御が十分でないとアクセス権限のないファイルの読み書きや任意のプログラムの実行が可能となる．これによりデータの改竄・漏洩，システムの停止など様々な脅威が発生し得る．

Linux で提供されている監査 (auodit) 機能を使いセキュリティに関わるイベントをログに保存することができる．定型処理しているような IoT デバイスにおいては正常動作時のイベント列が定義できる可能性が高い．このような場合，人が判断せずともそのイベント列から逸脱した場合には不正が発生していることになり不正対応を自動化できる可能性がある．

3.3.3 脆弱性

OS カーネルの脆弱性を使い権限昇格が可能であると，その権限で行える処理を不正に実行することが可能となる．強制アクセス制御機構を用いて被害を限定させることはできる．

本 OS カーネルの脆弱性に到る攻撃はアプリケーション経由あるいは不正侵入後にダウンロードされたマルウェア経由で行われる．動作しているプロセスの挙動，遠隔保守時に動かしているコマンドやプログラムの挙動を監視することにより異常な挙動を検出してこのような脆弱性による攻撃を受ける前に防げる可能性がある．

3.4 通信層

前節までに述べられているように，IoT デバイスで生成されたデータは，エッジを通じて計算主体となるクラウド上へと転送される．同様に，アクチュエータのようなデバイスであれば，クラウドで計算された結果に基づいた次のアクションが通知される．そのため，通信層ではネットワーク境界を跨ぐクライアント (IoT デバイス) およびサーバ (クラウド・ブローカ) 間の通信について議論する．このネットワーク境界では，スイッチやルータといったデバイスによって 5 タプルベースのアクセス制御が必須となる．このような環境において考慮すべき脅威を 2 つ取り上げる：(1) (D)DoS 等のデバイスへの攻撃 (可用性), (2) 中間者攻撃によるなりすましやトラフィック漏洩 (完全性, 可用性).

DDoS 等の攻撃は，クライアント (IoT デバイス) およびサーバ (クラウド) の両者で起こりうる．IoT デバイス自身の計算能力・ネットワーク性能ではローカルネットワークで生じる DDoS に対処することは一般に困難であり，既存のホストベースのアクセス制限によるトラフィック制御を考慮するのみである．ローカルネットワーク外からの DDoS 攻撃等は物理ネットワークにおけるネットワーク境界に位置するファイアウォールやルータ・スイッチによるフィルタリングが必要となる．同様に，サーバ側でのトラフィック制御は，ネットワーク境界でのアクセス制御を前提となることから，サーバに対する DDoS においては，既存のネットワークシステムにおける DDoS 緩和のフレームワークが適用可能である．ただし，多数の IoT デバイスが頻繁に接続されるような IoT ネットワークにおいては，ネットワーク境界でのアクセスリストの動的かつ高速な変更が必要となるため，粒度の細かい高速な制御が必要となる．同様に，アクセス制御の正しさを変更前に検証することで，より安全なネットワークを構築可能となる．

中間者攻撃による通信傍受やセッションの乗っ取りは既存のインターネットでも大きな問題である．例えば，IoT デバイスが接続されている有線・無線 LAN 内での通信はプロトコル・アプリケーションレベルでの暗号化が行われていない場合には傍受可能である．また，認証されていないエンド間の通信はセッション乗っ取りの可能性があり，送受信データの改竄やなりすましと言った脅威となる．これらの問題の解決には，エンド間の通信を暗号化し認証する必要がある．

3.5 物理／センシング層

センサ／物理層における特徴的なセキュリティ上の脅威として，IoT デバイスを物理的に窃取する Node Capture や，シャットダウン直後にメモリを窃取し，秘密鍵等の秘匿情報が漏洩する Cold Boot Attack のような脅威が起こりうる．これらは，機密性，完全性，可用性のすべてを侵

害する．

この他，機密性を侵害する脅威には，IoT デバイス外部から消費電力や電磁波等を観察し，機密データを漏洩させる Side-Channel Attack (SCA) がある．完全性を侵害する脅威には，不正侵入やメモリカードの書き換え等による，Malicious Code Ingection，Reprogramming，False data Injection，センサへの物理的な攻撃によりセンサデータを改竄する Jamming がある．可用性の観点では，電力を故意に消費させる Sleep Deprivation や停電，物理的な破壊等の脅威がある．

これらのセンサ／物理層の脅威への対策は，IoT デバイスやメモリカードを盗難されないように防御する，電源ボタンを押せないように保護する，IoT デバイスの挙動を観測されないようにする，耐タンパー性のある機器を利用するなど，物理的な対策を取る必要がある．本稿では，システムソフトウェアを研究対象としているため，センサ／物理層へのセキュリティ対策についてはスコープ外とする．

4. IoT におけるトラストワージネス

IoT のアーキテクチャは標準化団体 ISO/IEC JTC 1/SC 41 で議論されており，機密性，完全性，可用性からなるセキュリティの概念を IoT 向けに拡張したトラストワージネス（Trustworthiness）と呼ぶ性質を重視している [27]．日本から提案された ISO/IEC 30147:2021[43] は，一般的なシステムライフサイクルプロセスを IoT システムにおけるトラストワージネスの実装・保守のために拡張したシステムライフサイクルプロセスを定義している．トラストワージネスとは，標準的なセキュリティにプライバシ（Privacy），セーフティ（Safety），信頼性（Reliability），そしてレジリエンス（Resilience）を追加していることが特徴である．一般の IT では，セキュリティ 3 要素の中でも機密性が重要であるが，機密情報はいったん漏洩すると，その被害を食い止めるためのレジリエンス方策は限定的である．IoT デバイスが蓄積する機密情報は限定的である一方で，システムが停止しないように可用性を保つ行為はレジリエンスとして重要である．IoT のトラストワージネスを実現するには，侵害を受けないように耐タンパー性，認証，アクセス制御などのセキュリティ機能や信頼性を高めるだけでなく，侵害を検出し，被害を最小限に留め，その記録を残し，以後の改善に活用するレジリエンス機能にも重要な価値がある．本節では，IoT のトラストワージネスの実現に向けた課題として，特に信頼性とレジリエンスについて議論する．プライバシ，セーフティについては，応用事例に依存するため，ここでは割愛する．

4.1 未知のデバイスとの連携

現状の IoT は、実環境で動作する IoT デバイスがセンスしたデータをクラウドに収集する。たとえば、スマートメータは、家庭やオフィスの電力、ガス、水道などの使用量をあらかじめ設定されたクラウドに送信する。また、自動車のテレマティクスサービスは、メーカーが運用するクラウドサーバーに位置情報を送り、道路状況の情報を受け取ったり、メインテナンスサービスを実行する。将来の IoT デバイスは、互いに情報を交わすことでより高度なサービスを実現しようとするだろう。電力使用量を監視するスマートメータからの情報を受けて、エアコンやヒーターは、過大な動作を避けて省エネサービスを提供するかもしれない。初期には、これらの情報はクラウドを経由するであろうが、車車間通信のようなリアルタイム性の高い連携には、たとえば急ブレーキをかけたらその情報をすぐに後続車に送って衝突を避けるような制御も行いたくなる。

このような柔軟な連携を実現しようとすると、接続してくるデバイスが、信頼の置けるデバイスかどうかをリアルタイムで確認する認証が必要になる。この認証には、通常の IT システムでは、パスワードや証明書（PKI)、あるいは FIDO のような公開鍵を用いた認証が使われる。しかし、IoT においては、デバイスのすり替えやパスワードや証明書の秘密鍵を保存したメモリのコピーが比較的簡単に行えるので、これらの既存の認証法ではなりすましが行われやすい。

デバイスを安全に認証する方法には、たとえば PUF やデバイスのアナログ特性のようなハードウェア的な指紋に当たる情報を使ったり、デバイスが置かれた周囲の状況＝コンテキストから認証する方法などが提案されているが、(1) 真正デバイスの登録情報データベースが必要であり、クラウドへのアクセスが生じる、(2) コンテキストを偽装することで確実性が下がるなどの問題がある。未知のデバイスが信頼できるかどうかをその場で検証する方法が求められる。

4.2 ソフトウェアの信頼性とレジリエンス

IoT デバイスやエッジ，クラウド上で実行されるソフトウェアの信頼性低下もセキュリティ上の脅威である．開発者や配布元をなりすましたソフトウェア，または改竄されたソフトウェアが流通することにより，マルウェアを含むソフトウェア等の不正なソフトウェアが IoT 環境上で実行され，データ漏洩やデータ改竄につながる可能性がある．

また，インストール時は安全でも，その後脆弱性が発見されたソフトウェアを使い続けることでも，同様の脅威が発生する．IoT デバイスに配備されるソフトウェアの依存関係やバージョンを管理し，脆弱性が発見された際に対象となる IoT デバイスに対して適切かつ自動的にソフトウェアをアップデートする仕組みが必要である．

4.3 未対策の脅威に対するレジリエンス

IoT システムでは様々な対策困難な脅威が考えられる．

例えばIoTシステムではIoTデバイスが直接攻撃の対象となり，物理的に搾取される可能性がある（3.5節）．またソフトウェア上の脆弱性についても，その対策が広く知れ渡る前に攻撃者によって悪用される可能性がある（ゼロデイ攻撃）．さらにIoTシステムでは安定したネットワークが利用できるとは限らず，IoTデバイス上で実行されるソフトウェアを頻繁に更新できず，既知の脆弱性に対して対策が取られないままシステムが動作し続けるようなケースが想定される．このような脅威の下IoTシステム全体のレジリエンスを確保するためには，攻撃を受けた可能性のあるIoTデバイスを適切に発見・隔離する方法が必要となる．

4.4 アカウンタビリティ

一般に，アカウンタビリティを確保するためには，システムのログを改竄困難な形で保存できることだけでなく，侵害が発生した際に，そのログを解析することで侵害の原因を突き止められるようになっていることが求められる．侵害の原因を特定するためにはIoTデバイスの認証情報や付与されたアクセス制御情報の他に，関連する通信データやシステムコール，あるいはアプリケーション固有の情報などを適切に監視する必要がある．高いアカウンタビリティを保証するためには，これらの監視対象を人がアドホックに定めるのではなく，ある一定の基準に基づき系統的に定める方法が求められる．また多数のデバイスが参加することが想定されるIoTシステムでは，監視情報の質を保ちつつデバイス数に対してスケールする形で管理することが求められる．

5. ゼロトラストIoTアーキテクチャの実現に向けたシステムソフトウェア研究課題

ゼロトラストアーキテクチャは，以下の7原則に基づき設計，配備される[3]．

(1) 全てのデータとサービスはリソースと考える．
(2) 全ての通信はデバイスによらず保護される．
(3) リソースアクセスはセッション単位で動的に許可される．
(4) リソースアクセスポリシは動的に設定される．
(5) 保有機器の完全性とセキュリティ体制は監視・評価される．
(6) リソースアクセス前に認証&承認する．
(7) 保有機器，ネットワーク機器，通信の状態を把握しセキュリティ体制を改善する．

本稿では，3節で述べたIoTシステムの各階層におけるセキュリティ上の脅威への対策と，4節で述べたトラストワージネスの実現のため，ゼロトラストアーキテクチャの原則をIoTに拡張する．特に，通常のゼロトラスト実装[2], [4]では解決できない，IoTデバイスからエッジにおける通信層からミドルウェア層のシステムソフトウェア研究に着目する．

表1 3節IoTシステム各階層のセキュリティ上の脅威，4節トラストワージネスへの課題と，5節研究課題の対応．

IoTシステムの課題	ZT-IoTシステムソフトウェア研究課題
3.1 アプリ層	スコープ外． ただしポリシ決定はアプリ層で行う．
3.2 ミドルウェア層	5.4 監視，5.5 アクセス制御， 5.6 トラストチェーン
3.3OS 層	5.1 RoT，5.2 セキュアブート， 5.3 TEE，5.4 監視，5.5 アクセス制御
3.4 通信	5.4 監視，5.5 アクセス制御
3.1 センサ／物理層	スコープ外
4.1 デバイス連携	5.6 トラストチェーン
4.2 SW の信頼性	5.7 ソフトウェアの信頼性保証 5.8 ソフトウェアのライフサイクル管理
4.3 レジリエンス	5.9 形式検証によるレジリエンス
4.4 アカウンタビリティ	5.10 形式検証によるアカウンタビリティ

図4に，我々が想定するゼロトラストIoT（ZT-IoT）アーキテクチャの概要を示す．図では，IoTデバイス，エッジ／フォグ，クラウドからなるIoTシステムの実装例に対して，我々の提案するZT-IoTシステムソフトウェアとセキュリティコントローラ，および認証局等の信頼技術により，トラストチェーンを継続的に築いていく．ZT-IoTシステム・ソフトウェアでは，IoTデバイスやエッジの処理，およびIoTデバイスとIoTデバイス，IoTデバイスとエッジといったリソース間の通信は予め決められたセキュリティポリシに基づき実行される．

IoTデバイスやエッジで動作しているプロセスの挙動はそれぞれの機器で動作している監視デーモンにより監視される．監視デーモンは，正常遷移から逸脱するプロセスを発見するとセキュリティコントローラに報告する．セキュリティコントローラは，セキュリティポリシエンジンで監視デーモンから報告された異常状態を評価する．なんらかのポリシ違反や新たな脆弱性等が発見された場合には，ポリシで決められたアクションに従い，ホスト証明書の無効化やアクセスコントロールリストの更新等，対処法をIoTデバイス，エッジ，クラウドや認証局に指示する．

以降では，ZT-IoTシステムソフトウェアで取り組むべき課題と関連する技術について議論する．**表1**は，3節のIoTシステム各階層におけるセキュリティ上の脅威，4節トラストワージネスへの課題と，本節で述べる研究課題の対応を示したものである．

5.1 Roots of Trust

米国NIST[44]において，Roots of Trust (RoTs) は特に重要なセキュリティ機能を実行するハードウェア (HW)，ファームェア (FW) およびソフトウェア (SW) を統合したコンポーネントである．コンポーネントは本質的に信

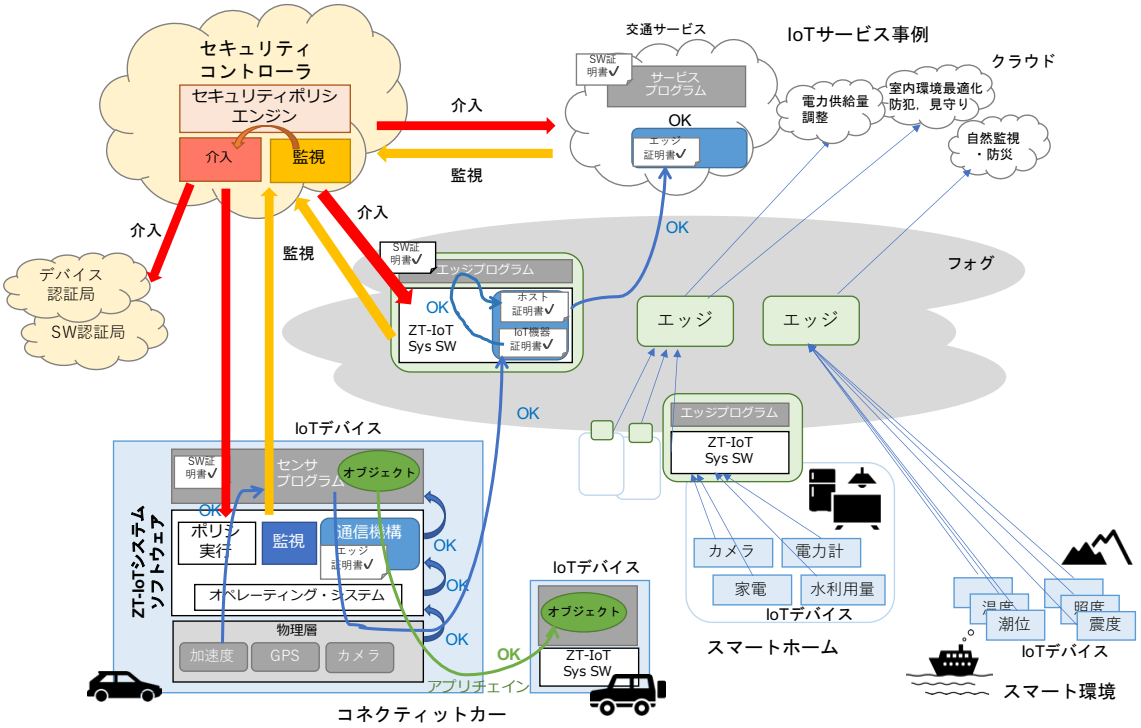

図 4　ゼロトラスト IoT（ZT-IoT）アーキテクチャの概要.

頼される事が要求され，高度に安全性が保障されなければならない，としている．このコンポーネントを Secure Element(SE) と称する．

SE が本質的に信頼されるには，暗号化／復号に利用する鍵等の保護すべき情報や機能を内包した SE が，攻撃に対する耐性として耐タンパー性を有し，安全なサプライチェーンによる SE 設計，SE 開発，SE 製造，SE 書込み，SE 配送，SE 利用（製品組立，製品配送，製品利用，製品廃棄），SE 破棄が実施されトレース可能であり，必要に応じて参照もしくは公開される事である．SE もしくは製品利用者は自身が使う SE がどのような成り立ちで製造され配送されたかを確認することで信頼する．しかし，このようなトレーサビリティは現実には無い．そのため利用者は製造者を信頼して利用しているのが現状である．よって製造者自身による故意もしくは過失による仕様以外の性能や機能が SE に混入した時，利用者が発見することは困難である．

本質的な信頼が無い状態は課題であると考えるが，本稿では市販されている SE は本質的に信頼したものとして扱う．SE 製品には Trusted Platform Module (TPM)，Yubico YubiHSM，Renesas Trusted Secure IP (TSIP)，Google Titan，Apple T2 等がある．

RoTs に必要な特に重要なセキュリティ機能数，利用方法は SE 製品により異なる．主な機能は SE 内部で秘匿され改竄不可能な鍵 (Hidden Root Key:HRK)，真性乱数生成器，暗号化／復号器，hash 生成器，セキュアストレージと各機能を利用する Application Programming Interface (API) である．

SE の利用例としてはセキュアブートや Microsoft 社の Bitlocker 等が上げられる．セキュアブートでは SE 内に保管した構成情報を元にした改竄の有無を計測しながらシステムをブートする．BitLocker では構成情報と暗号化／復号するための鍵保管に SE を利用する．

4.1 節で述べたように IoT デバイスは秘密鍵が漏洩するリスクが高い，IT システムにおいても秘密鍵が安全に管理してない状態では同様である．安全に秘密鍵を管理するために SE を利用する．しかし，SE 単体だけでは安全とは言えない．Zero Trust 空間ではどのようなタイミングや場所であっても成りすましや改竄等の攻撃が成立しえるからである．このような状況に対応する為に例えば，TPM は Attestation(構成証明) を保持して構成を測定した結果を第 3 者が検証する仕組である Remote Attestation (RA) 利用して定期的に確認することで回避する．

RA に対する攻撃は Trusted Execution Environment (TEE) により RA を実行する環境を普段使う環境と隔離することで攻撃が成立するリスクを低減する可能性が見込めるが，今後実証は必要である．

5.2　セキュアブート

セキュアブートとは Intel や ARM では以下の様に述べられている．

- Intel - セキュアブートとは？ [45]
  セキュアブートとは、最新の統合型拡張ファームウェアインターフェイス (UEFI) 2.3.1 仕様 (正誤表 C) の

機能の 1 つです．セキュアブートでは，デジタル署名を検証することで，ブートローダー，主要なオペレーティング・システム・ファイル，および不正なオプション Rom による改ざんが検出されます．

- ARM - Arm Platform Security Architecture Trusted Boot and Firmware Update[46]
  Serure Boot は Boot プロセスにおける各ステージでコード実行が許可されているか確認後に実行する．

利用する CPU，SoC，SE によりセキュアブート実装は異なるが，共通するのは以下の点である．

- boot 時にコードを検証して実行の可否を判断する．コードは 5.1 で述べた HRK から信頼を連鎖させられる鍵を用いて検証し，改竄を検知する．
- Boot 時に SE を利用する部分，検証，検知するコードは Read Only な場所に保存する．

IoT デバイスで利用される SoC はセキュアブートが利用するコードを安全に保管するメモリ若しくはストレージ確保が困難な場合が多く課題となる．

5.3 TEE

TEE は Multi Core CPU が持つハードウェアを利用して一部 Core を通常の実行環境 (Normal World:NorW) から切り離した実行環境 (Secure World:SecW) である．例えば，ARM TrustZone (ARM-TZ)，RISC-V Keystone，AMD SEV や Intel SGX 等である．

本稿でターゲットとしている IoT デバイスで多く使われている Arm A-Profile architectures における ARM-TZ ではブート時に Trusted Firmware (TF) を RAM 読込→実行することで TEE が利用出来る状態になる．次に SecW 内で利用可能なペリフェラル・インターフェースを Device Tree Blob (DTB) により決定し，SecW で実行するプログラムを RAM に読込む．SecW 内で実行するプログラムは Trusted Application (TA) と称されており，TA は NorW から隔離され改竄や追加が困難な状態となる．DTB，TA の RAM 読込先は System on Chip (SoC)，BootROM 内のプログラムや HW 設計により異なるが，NorW からはアクセス出来ないように制御されている．

ここで疑問なのが TF や TA が本物か，もしくは脆弱性や不具合が無いのか，という点である．TEE はあくまでも実行環境である．正規の手順をたどって構成された SecW で，正規な手順で呼ばれた TA は必ず実行する．よって真偽判定，その他検証は RAM 読込以前に実施しなければならない．つまりブートに組込む必要があり，Secure ブートを用いて検証しながら RAM 読込→実行することで改竄等のリスクが低減可能である。

しかし 5.2 で述べた課題があり FAT ファイルシステム等に保存された 2nd もしくは 3rd boot loader にセキュアブートを組込みこんで TF，TA 等を RAM 読込む方法をとることが多い．例えば，Broadcom BCM2837 SoC を使った Raspberry PI 3B+ (RPi3B+) では電源 ON にしてから VideoCore が BootROM 内のプログラムを実行してから/boot/start.elf が/boot/config.txt を読み込んだ後に TF と DTB を RAM 読込→起動することで TEE が利用可能となるが、ここに S セキュアブートを組み込むには TF に含まれる BL1.bin，BL2.bin，BL3.bin を改良してセキュアブート機能を埋め込んだうえで、物理的に交換が艱難なストレージ・デバイスに ReadOnly ファイルシステムを作成して配置しなければならない．SecW が利用可能になるまでに読込まれる bootcode.bin，loader.bin，start.elf は非公開かつ FAT 等で format されたファイルシステムから読込まれる．この方法だとファイルシステム上で TF が改竄される可能性は高く課題である．

5.1 で述べた SE との連携にも課題がある．ハードウェア的な課題として例えば，多くの SoC は CPU と SE が分離されており SPI や I^2C 等で結線されるが，配線が露出しており盗聴などが成立する可能性が高い．物理的に入れ替えることも難しくない．ソフトウェア的には SecW における SE の利用はまだ発展途上であり安全に利用するためには継続した調査，研究が必要である．

TEE を利用したとしても動作するのは HW であり SW である．将来において脆弱性が発生しえない HW/SW を開発することは不可能である．よって動作や通信を監視して調査，攻撃，脆弱性等の有無を確認することが重要である．

5.4 監視

5.4.1 OS/ミドルウェア層における監視

3 節で述べたとおり，改竄されたアプリケーションの実行，マルウェアの実行，不正侵入は，実行中のプロセスの挙動やシステム状態を監視することにより検知できる．プロセスの挙動は，プロセスが発行するシステムコールを監視することで把握できる．システムの状態は，CPU 資源，メモリ資源，ネットワーク資源などの利用状況を監視することにより把握できる．サーバ系システムではこれら情報をログファイルに蓄積するあるいはログをリモートマシンに送信し，ログを解析することによりシステムの異常を検知する．IoT デバイスでは機器内でログを解析するだけの CPU パワーがあるとは限らない．また IoT デバイスが外部サーバにログを送信するのはネットワーク負荷を考えると現実的でない．

我々は以下のアプローチで監視システムの実現を検討している．プロセスが正常動作している時のシステムコール発行遷移（ここでは正常遷移と呼ぶ）をあらかじめ定義する．監視デーモンはプロセスが発行するシステムソフトウェアを監視し，正常遷移と同値の遷移をしているかを実時間で検証する．

正常遷移から逸脱しているプロセスの状態は，その時のシステムコールの種類から，アプリケーションが改竄されているか，マルウェアが実行されようとしているか，不正侵入の可能性があるか，あるいは疑陽性 (false-positive) である．監視デーモンはプロセスの状態ならびに IoT デバイスの CPU 負荷やメモリ使用量などの資源利用状況をセキュリティコントローラに報告する．

なお．監視対象プロセスはアプリケーションプロセス．telnet や ssh などのシステムデーモンプロセス．監視プロセスそのものも含まれる．

5.4.2 通信層における監視

通信における正常・異常な振る舞いをいかにして検出するには，ネットワークでのモニタリングが必要不可欠である．デバイスの振る舞いは形式的手法による検証が可能であるため，通信が設計時に想定した仕様どおりであるかを検証することができる．しかし，全ての振る舞いの仕様を書き下すことは簡単でなく，想定外の動作が生じる可能性はある．通信パターンの異常を検出することができれば，その異常をセキュリティコントローラ等に通知することにより，さらなる検証や制御の詳細化が可能となると期待できる．現在主流となりつつあるデータ転送プロトコル (QUIC, TCP/TLS) では暗号化された通信が標準であることから，DPI 等の従来のパケットキャプチャによるペイロード解析によるモニタリングを行うことは困難である．そのため，フローごとのトラフィックパターンの変化より潜在的な異常を検出する必要がある．

現実的には，ネットワーク境界面でのスイッチやルータにおいて，(サンプリング) Netflow/Sflow 等のフローデータ収集により，タイムスタンプつきのフローレベルでの統計情報を監視するアプローチが挙げられる．同様に，ブローカ等のデータ集約点においてフロー情報を監視することも可能である．どちらの場合においても，大規模 IoT システムでは多量のデータが集まるため，何らかの集約方法およびそれに適した正常・異常検出手法が必要となる．

5.5 アクセス制御

5.5.1 OS/ミドルウェア層におけるアクセス制御

図 3 で挙げた通信プロトコルでは，DDS はアクセス制御の仕様が規定されているがそれ以外のプロトコルではアクセス制御の仕様が標準化されていない．MQTT の実装である mosquitto[47] や Apache Kafka はアクセス制御を記述できる．

ミドルウェアレベルで通信を監視し，ACL を実施する仕組みとして Envoy[48] プロキシがあり，エッジへの適用を検討している．Envoy は，メッセージブローカのフロントエンドプロキシとして配備することが可能であり，メッセージブローカの実装の 1 つである Kafka への通信監視を行う Kafka Broker filter も実装されている．Envoy を IoT デバイスとの TLS 通信のエンドポイントとして利用すれば，通信データを復号して監視したり，適宜アクセス制御を行うことが可能になる．また，メッセージプロキシやエッジサーバの実装に関わらず，共通の認証認可の仕組みを提供することもできると考えている．

IoT デバイス上で制御・サービス系アプリケーションが動いてる場合，IoT デバイス上でもアクセス制御する必要がある．IoT デバイス上で Envoy プロキシのオーバヘッドを評価し必要とするハードウェア要件を明らかにする．また，Envoy プロキシを導入しなくても IoT デバイス上でアクセス制御機構を実現する計画である．

通信プロトコルのアクセス制御仕様あるいは実装に依存して実現されているアクセス制御とは独立にアクセス制御インターフェイス (IACL: Independent Access Control List) を提供することにより，通信プロトコルの違いを意識せずに IoT デバイスのセキュリティポリシを記述可能となる．アクセス制御の記述はセキュリティポリシの性質を理論的に解析できるよう仕様を決める．

通信ライブラリの実装毎に IACL を組み込むのは開発および保守の観点から問題がある．通信層で通信データに含まれているオペレーションを解析できれば，通信ライブラリに直接変更を加えずに IACL を実装することが可能である．しかし，通信層の通信データは SSL で暗号化されており通信データに含まれているオペレーションを知ることが出来ない．そこで，アクセス制御機構が通信データを読めるように，SSL ライブラリ関数が呼び出される時にフックできる仕組みを導入することを検討している．

3.3 節で述べたとおり，リモート保守のための telnet や ssh から不正アクセスされる場合がある．アカウント毎にログインシェル上で実行可能なコマンドやアクセスできるファイルを制限することによりマルウェアが仕掛けられないよう防御することができる．Linux カーネルでは SELinux や Tomoyo のような強制アクセス制御 (Mandatory Access Control) 機構を利用できる．保守時に ACL ファイルを更新する必要が生じる場合．その更新が正当な権限を有するものから行われ偽装者からの更新でないことを保証する必要がある．

5.5.2 通信層におけるアクセス制御

前章で述べたように、通信におけるアクセス制御はサーバ・クライアント側で考慮する必要があるが、以下ではサーバサイドのみを議論の対象とする。サーバ側への攻撃には、オープンフロー等に代表される SDN によるネットワークスイッチおよびコントローラから構成されるネットワーク境界での制御となる。既存のファイアウオールベースのネットワークとゼロトラストアーキテクチャの異なる点は，恒久的な VPN 等のトンネルによるアクセスを許さず，ネットワークポリシに基づいた比較的短時間・細粒度でのアクセス制御が必要になる点である．そのため，ク

ライアントからのアクセスリクエスト，認証局・ポリシーサーバでの認証，境界面での ACL の変更という一連の手続きの繰り返しとなる．このような動的変更を伴うネットワークのアクセス制限の実現には，SDN によるアプローチが親和性が高く多くの研究開発が行われている．

我々は以下のアプローチで境界面のスイッチに投入される設定の妥当性を担保していくことを考えている。既存のエンタープライズネットワークでも設定投入時の間違いにより多くのネットワーク障害が生じており，ポリシが正しく実装されるかを検証する手法が必要なる．アプリケーションレベルでのポリシが最終的にネットワークフロー (5 タプル) の ACL へとマッピングされることから，ポリシから ACL へのマッピングの正しさ，およびネットワーク境界面全体として見た場合にポリシが充足されているかを検証することになる．さらに実際に投入された設定がデータプレーンのレベルで正しく動作しているかを検証することも必要になる．これらの問題を解決するために，本課題では，ゼロトラストネットワークに適した，コントロールプレーン・データプレーンにおけるネットワーク検証に関する研究を進める．既存のネットワーク検証では，比較的変更の少ない安定的なネットワークを暗黙的に仮定しているが，ACL の動的な変更を検証するためには，高速かつ大規模な検証を行うことが求められる．

5.6 トラストチェーン

5.6.1 IoT デバイスのリアルタイムトラスト

4.1 で述べたような IoT デバイス間で相手デバイスの信頼性を検査する方法を検討する．問題の根底には，相手デバイスのシステムプログラムすら信用できないことがある．PC やサーバーでは，通信相手の OS やプロトコルスタックまでを疑うことはあまりしない．Web サーバやブラウザを偽装して，情報を収集することはありうるが，これらはユーザープロセスである．システムプログラムが改造されないように，Windows10 以降，セキュアブートの使用が一般化している．セキュアブートは，SGX のような TEE によってもより堅固に実現される．Administrator 権限や TEE によるセキュア環境の分離が有効なのは，これらが一般ユーザーのアクセスから隔離されているからである．

ところが，IoT デバイスでは，これらの隔離が必ずしも信頼できない．たとえば，多くの IoT デバイスは，Administrator だけが定義されており，すべてのユーザープロセスが Administrator 権限で実行されている．IoT デバイスの OS は比較的小規模で，攻撃者が手を入れることも可能である．そして，熟達した攻撃者であれば，たとえ TEE のプログラムであっても，書き換えることは可能である．つまり，我々は，接続してくるデバイスには，信頼できるコードは一つもないことを想定する必要がある．かろうじて信頼できる部分があるとしたら，それはセキュアエレメント（ハードウェア的な信頼の基点）に製造者が埋め込んだ秘密鍵である．この製造者の秘密鍵をよりどころとして，IoT デバイスの信頼のチェーンを築く必要がある．

現在，我々が意図しているのは，自デバイスから，相手デバイスの中に，相手デバイスの真正性を検査するコードを送り込むことである．相手デバイスからすると，送り込まれるプログラムコードは，マルウェアのようにも見えるので，マルウェアではないことを別途証明する必要も生じるであろう．

相手デバイスの真正性，信頼性が検査できたとして，次にデータの交換が行われる．相手デバイスのシステムチェックができたとしても，さまざまなユーザープログラムのすべての検査ができるわけではないので，個人情報など機微なデータを渡してあらゆる使用をさせることはできない．たとえば，暗号化したパスワードを送ることがあっても，その解読を許してはならない．このためには，プライベートなデータを隠蔽する機能を持つオブジェクトの仕組みが利用できると考えている．パスワードの例では，パスワードの正しさを検査するメソッドは公開しても，パスワードを取り出すメソッドは秘匿しておくことである．

5.6.2 ホスト間のリアルタイムトラスト

データを送受信するホスト間を信頼する仕組みとして，SSL/TLS サーバ証明書を用いた手法が用いられている．Web サイトでは，主要な認証局（CA）が発行したホスト証明書を用いることで，そのサイトの真正性が保証される．また，自己 CA で自己署名証明書を発行し，事前に信頼するホスト間で自己 CA 証明書を交換して，限られたグループ内での信頼通信に利用する場合もある．メッセージブローカの Kafka や Mosquitto でも後者のような利用方法が可能であり，IoT デバイスの自己 CA 証明書を予めブローカに登録しておくことで，ホスト間の信頼通信が可能になる．しかしながら，信頼されていたホストが不正侵入されたり，あるいはデータ漏洩により不正な IoT デバイスが詐称してブローカへのアクセスを試みる可能性もある．証明書が正しいものであれば，ホストで相手の不正を検知することは困難である．

ゼロトラスト IoT アーキテクチャでは，IoT デバイスやエッジから収集した監視情報をもとにセキュリティコントローラで不正なホストが発見すると，そのホスト証明書を速やかに失効させる必要がある．また，IoT デバイスやエッジ，クラウドのホストでは，通信相手の証明書が有効なものであるかどうか常に確認する必要がある．我々は，pub/sub 通信モデルを採用する複数ブローカ実装を抽象化する API を提供する IoT システムのためのライブラリ SINETStream[49], [50] を開発しており，SSL/TLS によるホスト認証および通信暗号化もサポートしている．SINETStream のレイヤで，ホスト証明書の有効性を適宜確認する機能を実装し，ゼロトラストにおけるホスト間信

頼通信を実現する．

5.7 ソフトウェアの信頼性保証

IoT 環境におけるデータ収集・解析等のために用いられるオープンソースソフトウェア等の安全な流通・利用を可能とする必要がある．

ソフトウェアの真正性を検証する技術としてコード署名が使われている．ソフトウェアの開発者・配布元が，ソフトウェア（コード）に署名するとともに，ソフトウェアの利用者が署名を検証可能とすることで，利用者が入手したソフトウェアの開発・配布元，ならびにソフトウェアに改竄のないことを確認できる．署名には，X.509 形式の電子証明書 [51] や OpenPGP[52] が用いられる．また，このフレームワークを実装するためのツール群である sigstore[53] では，ソフトウェアへのコード署名と検証に加え，利用者が安全に管理された署名ログを参照可能としている．TUF（The Update Framework）[54] はソフトウェア更新フレームワークのための仕様で，署名されたメタデータでコードの最新性保証することができ，次節で述べる複数の OTA 実装にも採用されている．コンテナイメージ向けの TUF の実装として Notary[55] がある．ただし，コード署名を受けたルートキットやランサムウェアも発見されていたり [56], [57]，コード署名の失効処理が正しく行われていないことも知られており [58]，コード署名による信頼性保証のみでは不十分である．

ソフトウェアの脆弱性診断ツールは，商用からオープンソースまで多数の言語に対応したものが開発されている．OWASP（Open Web Application Security Project）のサイトでは [59]，静的アプリケーションセキュリティテスティングツール（SAST, Stacis Application Security Testing）がまとめられている．SAST では，ソフトウェアの開発中に脆弱性を発見するため，開発効率の向上にも寄与する．コンテナイメージの脆弱性スキャンツールも複数開発されている [60], [61], [62]．コンテナイメージのスキャンツールは，一般にコードの中から問題となる箇所を発見するのではなく，既知の脆弱性を含むライブラリがコンテナイメージの中に含まれているかどうかで判断する．これらの技術により継続的に脆弱性診断を行い，脆弱性が発見された場合にはコード署名の失効を含むソフトウェアの利用を停止するための処理を早急に行う必要がある．

5.8 ソフトウェアのライフサイクル管理

コード署名のみでは，ソフトウェアの入手時点以降に脆弱性などの問題が発見された場合の安全性を保証することはできない．そのため，ソフトウェア入手後に問題が後から発見された場合に，速やかに当該ソフトウェアの利用を停止・更新するためのフレームワークが必要である．また，非常に多くの IoT デバイス上で動作するソフトウェアに対してこれを適用することのスケーラビリティの検証も必要である．

脆弱性が発見された際の対処として，ソフトウェアのライフサイクルを管理して自動的にインストール，アップデートを行う OTA（Over The Air）技術が必要不可欠である．ユーザ端末上で動作する OS やソフトウェアでは，新機能の追加や脆弱性を解消のためにクラウドのサーバからネットワークを介してソフトウェアをアップデートする仕組みとして OTA がすでに備わっている．自動車分野では，自動車のためのソフトウェアアップデート機能を提供する Uptane が提案され [63]，その参照実装も公開されている [64]．国内でも，多くの企業が参画している JASPAR（Japan Automotive Software Platform and Architecture）で，車載電子制御システムのソフトウェアやネットワークの標準化が進められている．小型の IoT デバイスのための OTA では，NervesHub[65] がある．Nerves は，Erlang VM 上で動作する関数型言語 Elixir を対象とした IoT 用ソフトウェアの開発，配備，管理を可能にするプラットフォームであり，NervesHub でファームウェアアップデート機能を提供している．GitHub Actions[66] のように，CI/CD（継続的インテグレーション／継続的デリバリー）機能とそれに対応するエージェントを用いることで，コンテナ単位で IoT デバイスの OTA を実現することも可能である．また，Linux の OTA アップデートを可能にするものとして SWUpdate[67] もある．

OTA では，OS ごとアップデートする方法と，差分コードのみをアップデートする方法がある．前者は送受信データサイズが大きくなる，再起動が必要で booting attack のような脅威にさらされたり，brick devide（文鎮化）となる可能性が高くなる．後者はそれらを解決できるものの，IoT デバイスごとにパッケージ管理が必要となる．これらの利点欠点を考慮しつつ，コード量の削減や，大量の IoT デバイスに対するスケーラビリティといった課題を解決する必要がある．

5.9 形式手法による未対策の脅威に対するレジリエンス

4.3 節で述べたように，IoT システムのレジリエンス確保のためには，攻撃を受けた可能性のある IoT デバイスを適切に発見・隔離する方法が必要となる．これに対し，我々は形式手法を用いた異常検知・隔離手法の開発を目指す．具体的には，正常系の仕様から，(1) 異常検知のために必要な情報や，(2) 異常が検知された場合に，隔離などのセキュリティコントローラーが取るべきアクションの決定に必要となる情報を導出する手法の開発を目指している．

5.10 形式手法によるアカウンタビリティ

4.4 節で述べたように，アカウンタビリティを確保するためにはシステムのログを解析し，侵害の原因を特定する

必要がある．IoT デバイスでは利用できる計算資源が乏しく，また様々な脅威に晒されることから，IoT デバイス上のログは随時クラウドへ送信することになる．クラウドへは多数のデバイスからログが送信されるため，その中から侵害に関係する情報だけを高速に抽出し保存することが求められる．また，IoT デバイスが動作する環境では必ずしも安定したネットワークが提供されているとは限らない．そのため通信量を削減するために IoT デバイスからクラウドへ送信するログを低コストでフィルタリングすることが求められる．今後はオートマトン理論を応用することで，抽出したい情報を簡便な形で指定でき，少ない計算資源で高速に動作するログ処理アルゴリズムの開発を計画している．また多数の IoT デバイスからログを受け取るクラウド側には大きな負荷がかかることが想定される．この負荷を事前に見積るために，クラウドシステム全体のスケーラビリティを検証する形式検証技術の開発を目指す．

さらに適切な監視対象を同定することもアカウンタビリティのための重要な課題である．今後は監視対象をアクセス制御情報やセキュリティポリシからの論理的帰結として定める手法，通信パターンなどのデータから統計的に定める手法，そしてそれらを融合する方法について研究し，監視対象を系統的に同定することを目指す．

6. おわりに

本稿では，ゼロトラスト IoT システムの実現に向け，システムソフトウェアにおける研究課題を示した．IoT デバイス，エッジ，クラウドの横方向の階層と，センサ／物理層，通信層，OS 層，ミドルウェア層，アプリケーション層の縦方向の階層を整理し，とくに IoT デバイスからエッジと通信層からミドルウェア層に焦点をあてて，IoT システムにおけるセキュリティ上の脅威とトラストワージネスの実現に向けた課題を整理した．また，それらに対して ZT-IoT の実現に向けた研究課題として，RoT，セキュアブート，TEE，監視，アクセス制御，トラストチェーン，ソフトウェアの信頼性保証とライフサイクル管理，および，形式手法による未対策の脅威に対するレジリエンスとアカウンタビリティについて議論した．

今後は，JST CREST「基礎理論とシステム基盤技術の融合による Society 5.0 のための基盤ソフトウェアの創出（S5 基盤ソフト）」領域，「形式検証とシステムソフトウェアの協働によるゼロトラスト IoT」プロジェクト [68] でこれらの研究課題に取り組んでいく．

謝辞 本稿の執筆にあたり貴重なご意見を頂いた田口研治氏, 小林久美子氏, 坂根栄作氏を始め，日頃より議論頂いている本 CREST プロジェクトメンバに感謝いたします．本研究は，JST，CREST，JPMJCR21M3 の支援を受けたものである．

参考文献

[1] 木村康則ほか：Society 5.0 時代の安心・安全・信頼を支える基盤ソフトウエア技術，技術報告，JST 研究開発戦略センター (2021).
[2] Ward, R. and Beyer, B.: BeyondCorp: A New Approach to Enterprise Security, *;login:*, Vol. Vol. 39, No. 6, pp. 6–11 (2014).
[3] Rose, S., Borchert, O., Mitchell, S. and Connelly, S.: Zero Trust Architecture, NIST SP 800-207, Technical report, NIST (2020).
[4] NCCoE: Implementing a Zero Trust Architecture, `https://www.nccoe.nist.gov/projects/implementing-zero-trust-architecture`.
[5] 後藤厚宏：戦略的イノベーション創造プログラム（SIP）第 2 期／ IoT 社会に対応したサイバー・フィジカル・セキュリティ，`https://www.nedo.go.jp/activities/ZZJP2_100123.html`.
[6] FIDO Alliance: FIDO, `https://fidoalliance.org/`.
[7] FIDO: FIDO Device Onboard Specification, `https://fidoalliance.org/specs/FDO/FIDO-Device-Onboard-RD-v1.0-20201202.html`.
[8] 内閣府：Society 5.0, `https://www8.cao.go.jp/cstp/society5_0/`.
[9] McKinsey Global Institute: The Internet of Things: Mapping the value beyond the hype, `https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/the-internet-of-things-the-value-of-\breakdigitizing-the-physical-world` (2015).
[10] Akpakwu, G. A., Silva, B. J., Hancke, G. P. and Abu-Mahfouz, A. M.: A Survey on 5G Networks for the Internet of Things: Communication Technologies and Challenges, *IEEE Access*, Vol. 6, pp. 3619–3647 (online), DOI: 10.1109/ACCESS.2017.2779844 (2018).
[11] Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P. and Sikdar, B.: A Survey on IoT Security: Application Areas, Security Threats, and Solution Architectures, *IEEE Access*, Vol. 7, pp. 82721–82743 (online), DOI: 10.1109/ACCESS.2019.2924045 (2019).
[12] 国立情報学研究所：SINET 広域データ収集基盤実証実験，`https://www.sinet.ad.jp/wadci`.
[13] Takasaki, C., Takefusa, A., Nakada, H. and Oguchi, M.: A Study of Action Recognition Using Pose Data Toward Distributed Processing Over Edge and Cloud, *2019 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 111–118 (2019).
[14] Fukushima, Y., Asai, Y., Aoki, S., Yonezawa, T. and Kawaguchi, N.: DigiMobot: Digital Twin for Human-Robot Collaboration in Indoor Environments, *2021 IEEE Intelligent Vehicles Symposium (IV)*, pp. 55–62 (online), DOI: 10.1109/IV48863.2021.9575499 (2021).
[15] 中島昭範（編）：「クルマ」と「モノ」をつなげる V2X 技術の動向と展望，Vol. 15, No. 2, 電子情報通信学会 通信ソサイエティマガジン (2021).
[16] Beckman, P., Sankaran, R., Catlett, C., Ferrier, N., Jacob, R. and Papka, M.: Waggle: An open sensor platform for edge computing, *2016 IEEE SENSORS*, pp. 1–3 (online), DOI: 10.1109/ICSENS.2016.7808975 (2016).
[17] Echigo, Y. and Ohtake, S.: Vibration Measurement of Signal Bonds for Shinkansen, *2021 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 1–5 (online), DOI: 10.1109/ICCE50685.2021.9427594 (2021).
[18] 牧野秀夫，中澤陽平，前田義信，高橋 昌，小林 真，若月大輔，井筒 潤，杉田 暁，福井弘道：準天頂衛星の補

強サービスと 2 アンテナ式 GPS を用いた歩行者移動経路の高精度測位に関する実証実験，電子情報通信学会技術研究報告, WIT2020-35，pp. 32–36 (2021).
[19] Machizawa, M. G., Lisi, G., Kanayama, N., Mizuochi, R., Makita, K., Sasaoka, T. and Yamawaki, S.: Quantification of anticipation of excitement with a three-axial model of emotion with EEG, *Journal of Neural Engineering*, Vol. 17, No. 3, pp. 1–17 (online), DOI: 10.1088/1741-2552/ab93b4 (2020).
[20] 近堂　徹，町澤まろ：脳生理情報のクラウド解析プラットフォーム実現に向けた通信手法の検討，Vol. 2021, No. 1, pp. 237–242 (2021).
[21] KDDI 総合研究所，KDDI，常葉大学，国土緑化推進機構：IoT を活用した森林管理効率化に関する実証実験を開始～シカなどによる森林被害を未然に防止～，https://www.kddi-research.jp/newsrelease/2019/082101.html (2019).
[22] Nakagawa, K., Shimotoku, D., Kawase, J. and Kobayashi, H.: Dependable Wildlife DTN: Wearable Animal Resource Optimization for Sustainable Long-Term Monitoring, *2020 IEEE 6th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (DependSys)*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 49–56 (online), DOI: 10.1109/DependSys51298.2020.00016 (2020).
[23] Zhao, W., Yang, S. and Luo, X.: On Threat Analysis of IoT-Based Systems: A Survey, *2020 IEEE International Conference on Smart Internet of Things (SmartIoT)*, IEEE, pp. 205–212 (online), DOI: 10.1109/SmartIoT49966.2020.00038 (2020).
[24] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S.: Fog Computing and Its Role in the Internet of Things, *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, New York, NY, USA, Association for Computing Machinery, p. 1316 (online), DOI: 10.1145/2342509.2342513 (2012).
[25] Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J. and Yang, X.: A Survey on the Edge Computing for the Internet of Things, *IEEE Access*, Vol. 6, pp. 6900–6919 (online), DOI: 10.1109/ACCESS.2017.2778504 (2018).
[26] Hu, Y. C., Patel, M., Sabella, D., Sprecher, N. and Young, V.: Mobile Edge Computing: A key technology towards 5G, Technical report, ETSI (2015).
[27] ISO/IEC: Internet of Things (IoT) — Reference Architecture, ISO/IEC 30141:2018，https://www.iso.org/standard/65695.html (2018).
[28] SecureWorld: Teen Finds Vulnerability That Can 'Annoy the Sh*t' Out of Tesla Owners, https://www.secureworld.io/industry-news/teen-vulnerability-annoy-tesla (2022).
[29] National Vulnerability Database: CVE-2022-23126, https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2022-23126 (2022).
[30] OASIS: MQTT Version 5 (2019).
[31] OMG: OMG Data Distribution Service (DDS) Version 1.4 (2015).
[32] Shelby, Z., Hartke, K. and Bormann, C.: The Constrained Application Protocol (CoAP), *RFC 7252*, IETF, (online), DOI: 10.17487/RFC7252 (2014).
[33] ISO/IEC: Information technology — Advanced Message Queuing Protocol (AMQP) v1.0 specification，*ISO/IEC 19464:2014* (2014).
[34] Kreps, J., Narkhede, N. and Rao, J.: Kafka : a Distributed Messaging System for Log Processing, *NetDB workshop 2011*, pp. 1–5 (2011).
[35] Shree, R., Choudhury, T., Gupta, S. C. and Kumar, P.: KAFKA: The modern platform for data management and analysis in big data domain, *2017 2nd International Conference on Telecommunication and Networks (TEL-NET)*, pp. 1–5 (online), DOI: 10.1109/TEL-NET.2017.8343593 (2017).
[36] Apache: Apache Kafka, https://kafka.apache.org/.
[37] W3C: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) (2007).
[38] P. Saint-Andre, E.: Extensible Messaging and Presence Protocol (XMPP): Core, *RFC 3920*, IETF (2004).
[39] Patrick Th. Eugster and Pascal A. Felber and Rachid Guerraoui and Anne-Marie Kermarrec: The many faces of publish/subscribe, *Computing Surveys*, Vol. 35, pp. 114–131 (2001).
[40] ONVIF: PTZ Service Specification Version 17.06, https://www.onvif.org/specs/srv/ptz/ONVIF-PTZ-Service-Spec-v1706.pdf?ccc393\&ccc393 (2017).
[41] Tim Dorr: Tesla JSON API, https://github.com/timdorr/tesla-api.
[42] New Jersey Cybersecurity & Communications Integration Cell: Mirai – NJCCIC Threat Profile.
[43] ISO/IEC: Information technology — Internet of things — Methodology for trustworthiness of IoT system/service, ISO/IEC 30147:2021，https://www.iso.org/standard/53267.html (2021).
[44] NIST: Roots of Trust at Glossary, https://csrc.nist.gov/glossary/term/roots_of_trust.
[45] INTEL: Secure Boot, https://www.intel.co.jp/content/www/jp/ja/support/articles/000006942/boards-and-kits/desktop-boards.html.
[46] ARM: Secure Boot, https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/PSA/DEN0072-PSA_TBFU_1.1-BETA0.pdf?revision=3ce2513a-ae0f-4b43-96a0-851ed67a640b&hash=5A93EC495294641ADD8880DA88619EC76D1743831.
[47] Eclipse Foundation: Eclipse Mosquitto, https://mosquitto.org/.
[48] Envoy: https://www.envoyproxy.io/.
[49] Takefusa, A., Sun, J., Fujiwara, I., Yoshida, H., Aida, K. and Pu, C.: SINETStream: Enabling Research IoT Applications with Portability, Security and Performance Requirements, *2021 IEEE 45th Annual Computers, Software, and Applications Conference (COMPSAC)*, Los Alamitos, CA, USA, IEEE Computer Society, pp. 482–492 (online), DOI: 10.1109/COMPSAC51774.2021.00073 (2021).
[50] 国立情報学研究所：SINETStream, https://nii-gakunin-cloud.github.io/sinetstream/.
[51] IETF: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, https://www.ietf.org/rfc/rfc5280.html.
[52] IETF: OpenPGP Message Format, https://www.ietf.org/rfc/rfc4880.html.
[53] The Linux Foundation: A new standard for signing, verifying and protecting software, https://www.sigstore.dev/.
[54] Cloud Native Computing Foundation (CNCF): The Update Framework (TUF), https://theupdateframework.io/.
[55] Docker: Content trust in Docker, https://docs.docker.com/engine/security/trust/.
[56] ISTRATE, C. A., BIRO, B., BLEOTU, R. C. and COBLI, C.: Digitally-Signed Rootkits

are Back A Look at FiveSys and Companions, https://www.bitdefender.com/blog/labs/digitally-signed-rootkitsare-back-a-look-\breakatfivesys-and-companions/ (2021).
[57] Trend Micro: New Yanluowang Ransomware Found to be Code-Signed, Terminates Database-Related Processes, https://www.trendmicro.com/en_us/research/21/l/yanluowang-ransomware-code-signed-terminates-\breakdatabase-processes.html (2021).
[58] Kim, D., Kwon, B. J., Kozák, K., Gates, C. and Dumitras, T.: The Broken Shield: Measuring Revocation Effectiveness in the Windows Code-Signing PKI, *27th USENIX Security Symposium (USENIX Security 18)*, Baltimore, MD, USENIX Association, pp. 851–868 (2018).
[59] OWASP (Open Web Application Security Project): Source Code Analysis Tools, https://owasp.org/www-community/Source_Code_Analysis_Tools.
[60] Clair: https://github.com/quay/clair.
[61] Docker: Vulnerability scanning for Docker local images, https://docs.docker.com/engine/scan/.
[62] Trivy: https://aquasecurity.github.io/trivy/.
[63] Kuppusamy, T. K., Brown, A., Awwad, S., McCoy, D., Bielawski, R., Mott, C., Lauzon, S., Weimerskirch, A. and Cappos, J.: Uptane: Securing Software Updates for Automobilesj, *14th ESCAR Europe 2016* (2016).
[64] Uptane: Secure Over-The-Air Updates for Ground Vehicles, https://github.com/uptane/.
[65] NervesHub: Manage OTA firmware updates across your fleet of devices, https://www.nerves-project.org/nerveshub.
[66] GitHub Actions: https://github.com/features/actions.
[67] SWUpdate: Software Update for Embedded Linux Devices, https://github.com/sbabic/swupdate.
[68] JST CREST: 形式検証とシステムソフトウェアの協働によるゼロトラスト IoT，https://zt-iot.nii.ac.jp/.