Vol.2022-OS-156 No.2
2022/7/27

Pub/Sub通信モデルを用いたセキュリティ監視機構の検討

石川 裕[1,2]　Yin Jie[1]　竹房 あつ子[1,2]　松井 俊浩[3]　小野 泰司[3,1]　合田 憲人[1,2]

概要：Linux が搭載されているインターネットに接続された IoT 機器を想定したサイバーセキュリティ対策のための監視機能及び制御機能を実現するシステムの実現手法を検討する．監視対象は Linux Audit が扱っているシステムコール発行イベントだけでなく，CPU やメモリ使用量などシステム状態の統計情報，ユーザレベル通信プロトコルの挙動イベントを採取する機能を検討する．実時間でセキュリティ対策のための監視データ解析が実現できるよう，採取される監視データ量を削減するフィルタ機能を導入する．監視情報およびフィルタ機能は複数のプロセスにより実現される．これらプロセス群を協調動作させるためのデータ通信や制御通信は，pub/sub 型モデルの 1 つである MQTT 通信インターフェイスを採用する．プロセス間で交換される通信データの種類を識別する topic 名を規定することにより，構成要素を容易に置換・追加することが可能となる．

1. はじめに

我々はゼロトラスト IoT システム (以降，ZT-IoT システムと呼ぶ) の実現に向けたシステムソフトウェアの研究を行っている [1]．ゼロトラストとは，米国 NIST(National Institue of Standards and Technology) が発行した NIST-SP800-207[2] の定義では，サイバー攻撃からの防御を静的なネットワークベースの境界線 (例えば VPN や Firewall) から，ユーザー，資産，リソースに焦点をあてたサイバーセキュリティパラダイムの集合としている．

我々が提案している ZT-IoT システムソフトウェアは概念的には以下の 3 つのコンポーネントから構成される (図 1)．

- Monitoring コンポーネント
  セキュリティ対策のためにアプリケーションや OS の挙動を監視する機構を実現するソフトウェア．
- Security Policy Contoroller コンポーネント
  監視した内容を解析しセキュリティポリシに基づき IoT 機器および関連する資産を守るためにプロセス停止やアクセス制御などの方策を決めるソフトウェア．
- Enforcement コンポーネント
  Security Policy Controller コンポーネントで決定した方策に従い，プロセスの実行を停止させたりファイルアクセスやネットワークアクセス制限などを実施するソフトウェア．

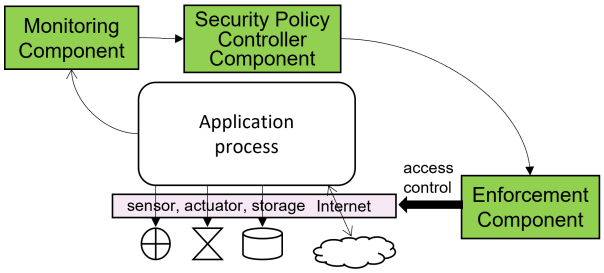

図 1　監視およびアクセス制御システム構成概念図

本論文では，Linux が稼働している IoT 機器を想定し，Monitoring コンポーネントであるセキュリティ監視機構の実現方法を検討する．セキュリティ監視機構の実現において我々が想定する IoT 機器と通常のサーバとの違いは次の 2 点である．i) IoT 機器はオペレータ不在で動作する，ii) IoT 機器に繋がっているネットワークは通信帯域が狭く IoT 機器が移動体の場合はネットワーク接続も不安定である．なお，IoT 機器は物理的に様々な人がアクセス可能な場所に置かれることが多く物理的脆弱性が多い．これによるセキュリティ対策のための監視機構についてはここでは扱わない．

我々は，上記 3 コンポーネントの機能を複数のプロセスによって実現することを検討している．本論文では，Monitoring コンポーネントの実装方法，ZT-IoT システムソフトウェアのプロセス群と協調動作させる枠組みを検討する．プロセス間のデータ通信および制御のための通信と

[1] 国立情報学研究所
National Institute of Informatics
[2] 総合研究大学院大学
The Graduate University for Advanced Studies (SOKENDAI)
[3] 情報セキュリティ大学院大学
Institute of Information Security

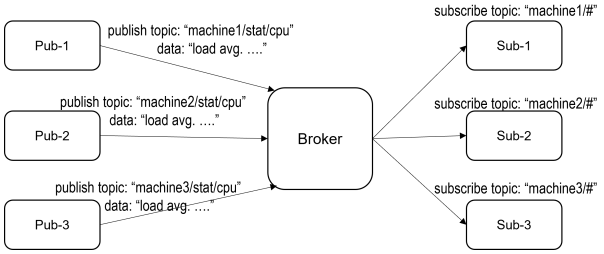

図 2 Pub/Sub 通信モデル

して pub/sub 通信モデルを実現している MQTT 通信 [3] の採用を検討する．MQTT 通信の概要は次節で紹介する．3 節で Monitoring コンポーネントに必要な機能を検討した後，実現方法を 4 節で検討する．関連研究・システムについて 5 節で述べた後，本論文をまとめる．

2. MQTT の概要

MQTT 通信 [3] は pub/sub 通信モデルの一つである [4]．図 2 に示すとおり，送信者 (Publisher) は受信者を特定せずメッセージに topic(キーワード (文字列)) を付与してブローカ (Broker) に送信する．Broker ではメッセージは topic 毎にグループ化される．受信者 (Subscriber) は topic を指定して購読 (Subscribe) 要求をブローカに送る．これ以降購読した topic が Broker に送られるとそのメッセージは受信者に送信される．

MQTT の topic 名は，Unix のファイルパス名のように”/” をつけて階層化出来る．また topic の指定にはワイルドカード文字 (“#”) が使用できる．図 2 では，sub-1 から sub-3 は，それぞれ，pub-1 から pub-3 の “stat/cpu”topic を受信する．図 3 に示す MQTT 使用例では，Pub-1 から Pub-2 送信者および Sub-1 受信者がのように topic 名でメッセージ送信あるいは購読要求している．

```
Pub-1:  publish topic: "ZT-IoT/m1/audit/syscall/pid1234"
                       data:"..."
        publish topic: "ZT-IoT/m1/proto/dds" data:"..."
        publish topic: "ZT-IoT/m1/net/traffic" data:"..."
Pub-2:  publish topic: "ZT-IoT/m2/audit/syscall" data:"..."
        publish topic: "ZT-IoT/m2/proto/dds" data:"..."
        publish topic: "ZT-IoT/m2/net/traffic" data:"..."
Pub-3:  publish topic: "ZT-IoT/m3/audit/syscall" data:"..."
        publish topic: "ZT-IoT/m3/proto/dds" data:"..."
        publish topic: "ZT-IoT/m3/net/traffic" data:"..."
Sub-1:  subscribe topic: "ZT-IoT/#/audit/#" data:"..."
Sub-2:  subscribe topic: "ZT-IoT/#/proto/dds" data:"..."
Sub-3:  subscribe topic: "ZT-IoT/#/net/traffic" data:"..."
```

図 3 MQTT 使用例

ここでは，最初の topic 階層が我々のシステム略称である ZT-IoT を使い，第 2 階層目が機器名，第 3 階層目が監視対象名，第 4 階層目は監視対象を細分化した名前を指定している．本例では，Sub-1，Sub-2，Sub-3 は，それぞれ，Publisher が指定した第 2 階層の machine ID に関係なく，audit，proto/dds，net/traffic に関するメッセージを受信する．

このように MQTT が提供する topic 名の階層構造および受信時のワイルドカード指定により選択的にメッセージを受信することが可能とある．

3. 監視対象対象とフィルタリング

本節で Monitoring コンポーネントに必要な機能を設計するにあたり重要となるサイバーセキュリティ監視対象を同定するとともに実時間でセキュリティ対策のための監視データ解析が実現できるよう採取される監視データ量を削減するフィルタ機能を検討する．

3.1 監視対象

監視対象はサイバー攻撃からどのような資産を守るのかによって異なってくるが，システムソフトウェアとしては計算機内で動作しているソフトウェアの挙動を可能な限り監視でき，必要な情報を選択出来る機構が必要である．我々は以下の監視対象を検討している．

(1) 監査データ
監査データは監視イベントの要因になったユーザ ID が紐付けられたイベントであると捉えることが出来る．ユーザプロセスが想定外の計算資源にアクセスしていないか，意図されていない相手と通信が行われていないか，想定されていないサイトからログインされていないか，などを監視する．このような監視システムとしては Linux Audit がある．
計算資源およびネットワークのアクセス制御により計算資源やリモートログインが限定されている環境においても，システムの脆弱性によりサイバー攻撃を受ける．監査データを実時間で解析することにより被害を最小限に留めるとともに，被害の同定が可能となる．また，既知の脆弱性により攻撃を仕掛けてくる場合には，監査データの実時間解析結果からそのネットワーク接続を切断し相手の通信をブロックすることが可能となる．

(2) システム統計情報
CPU 使用率，メモリ使用量，ネットワークトラフィック量を観測することにより，Linux Audit では得られないシステムの異常が検知可能となる．例えば，アプリケーションの脆弱性により CPU を大量に消費するプログラムがインジェクションされた場合を考える．正常状態での CPU 使用率が既知であれば定期的に CPU 使用率を監視することにより，システムの異常が検知でき，セキュリティ対策が可能となる．

(3) アプリケーションレベル通信プロトコル挙動
IoT 機器のアプリケーションが利用している通信プロトコルには，MQTT[3], DDS[5], CoAP[6], AMQP[7],

Apache Kafka[8], SOAP[9], XMPP[10], HTTP を使った Web API などがある．通信プロトコル実装の脆弱性によりアプリケーションを停止させられる可能性がある．例えば，2021 年に National Vulnerability Database に登録された CVE-2021-28166[11] は，MQTT 実装の一つである Eclipse Mosquitto の脆弱性である．通信プロトコルに違反したパケットを受信すると NULL Pointer アクセスで実行が停止する．サイバー攻撃に悪用される可能性のある脆弱性である．このような攻撃に対して通信データを監視しできれば，プロトコル解析によりプロトコル違反をしていないか検知できる．

3.2 監視データフィルタリング機能

セキュリティ対策のために無闇に監視対象を拡大すると監視データは膨大になる．監視しているアプリケーションの正常時の挙動遷移が既知であれば，その挙動遷移をしている限りは Security Policy Controller に監視データを送信する必要がない．

例えば，システム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．正常遷移から逸脱している状態にあるかは，その時のシステムコールの種類やシステムの統計情報から，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) であると判断できる．

監視データから正常遷移と同値の遷移をしているかを実時間で検証し，逸脱した場合にその遷移に至った監視データを Security Policy コンポーネントに送る．

4. 実現方法の検討

現在検討中のソフトウェア構成を図 4 に示す．図中，角丸四角形で囲まれたモジュールはプロセスとして実装される．Audit，Statistic Monitor は，それぞれ，前節で述べた監査データ，システム統計情報を採取するためのプロセスである．監視データフィルタリング機能は，Filter プロセスで実現される．アプリケーションレベル通信プロトコル挙動を監視するために Comm. P. Analyzer 動的ライブラリがアプリケーションプログラムに動的にリンクされる．実線矢印はプロセス間通信，破線矢印はプロセスあるいはカーネル内の実行フローを表現している．

IoT 機器内のプロセス群の通信・制御は MQTT 通信で実現される．MQTT 通信ライブラリが提供する topic 名の階層構造化を使って監視データの授受がおよび制御のためのメッセージ通信が容易に定義できる．詳細は，4.6 節で述べる．

図 4 には，Enforcement コンポーネントおよび Security Policy Controller コンポーネントへの矢印がある．Enforcement コンポーネントである Enforcement プロセスは Security Policy Controller や Filter，Comm. P. Analyser から通知される指示に従い，プロセスの停止，ネットワークコネクションの遮断，アクセス権限変更，などを行う．以下，Audit，Statistic Monitor，Comm. P. Analyser の実現方法を検討した後，Coordinator モジュールの実現方法を示す．監視機能に含まれる重要な要素であるフィルタリングプロセスの具体的実装は今後の課題である．

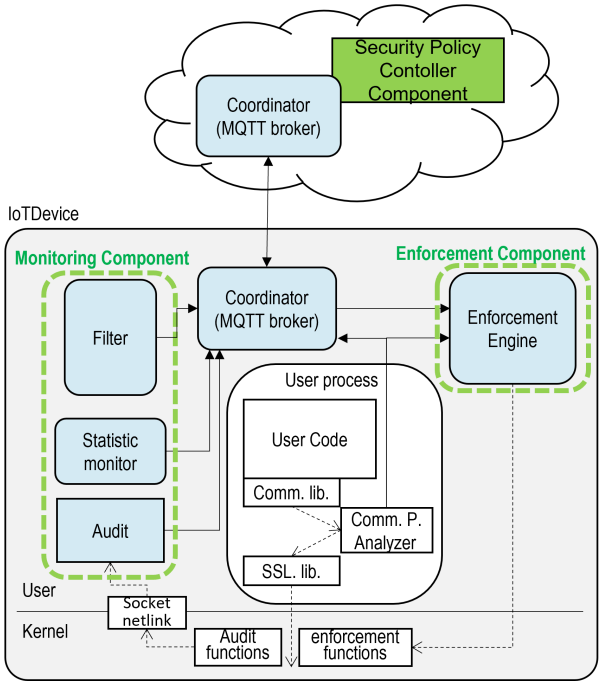

図 4 ソフトウェア構成

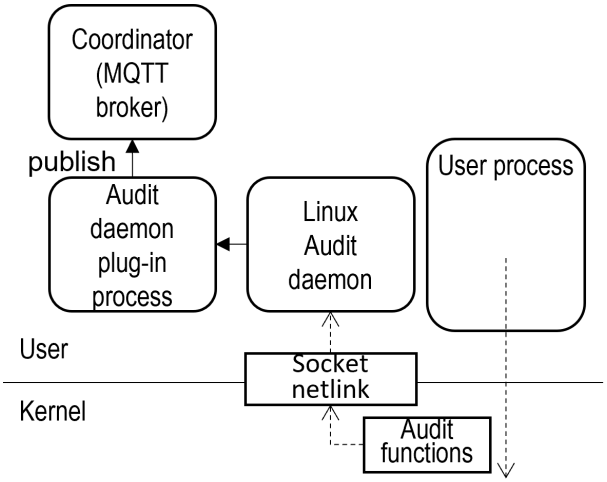

図 5 Audit 実装検討 (Plug-in)

4.1 Audit

Linux Audit が生成する監査データを利用する．Linux Audit はシステムコールの監視だけでなく，ログインや暗号鍵アクセスなどの監視が可能である[*1]．

[*1] Linux Audit が生成するメッセージタイプが

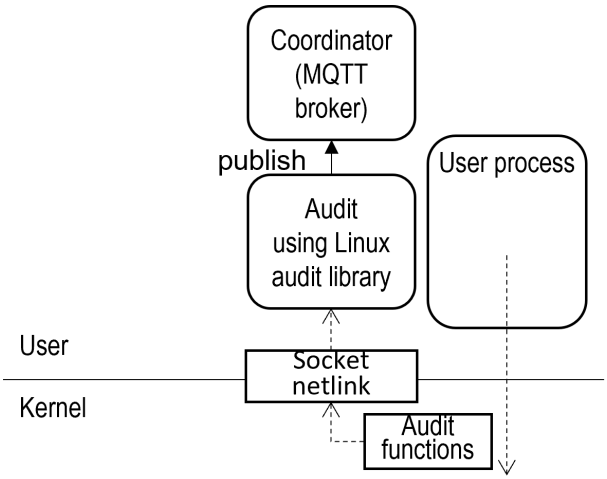

図 6 Audit 実装検討 (Linux audit library)

4.2 Auditd Plug-in による実装

図 5 は，Linux の Audit デーモンが提供する Plug-in 機能を用いた実装を示している．本 Plug-in 機能をデーモンプロセス内にユーザのロジック埋め込むのではなく Audit デーモンと通信するプロセス（Plug-in プロセス）を生成する仕組みである．Plug-in プロセスは標準入力から監査データを受信する．受信した監査データを Coordinator プロセスである MQTT Broker プロセスに送信 (publish) する．

一般に使われている Linux Audit デーモンは，ローカルディスクに監査データを保存するだけでなくリモートサーバに監査データを送信する Plug-in プロセスも提供されている．既に何らかの Plug-in プロセスを実装しているシステムでも我々のシステムが使われるように汎用性をもたせる場合にはこのアプローチを採用することになる．

4.3 audit ライブラリを使用した実装

図 6 は，Linux Audit が提供しているライブラリを使った実装を示している．Linux Audit デーモンが使っている Audit 機能のためのライブラリは公開されている．このライブラリは Linux Kernel で生成される監査データを netlink socket から読み込むライブラリである．受信した監査データは Plug-in プロセスと同じく Coordinator プロセスである MQTT Broker プロセスに送信 (publish) する．本アプローチは Audit Plug-in プロセスが介在しないためオーバヘッドが低減する．

Linux Audit の汎用性かオーバヘッドの最小化を目指すのかは，今後，プロトタイプ実装および評価を通して決めていく．

4.4 Statistic Monitor

Statistic monitor プロセスは/proc ファイルシステムで採取できる CPU，メモリ，ネットワーク利用状況を定期的に取得し，MQTT Broker にシステム利用状況を送信する．Audit プロセスと統合することも可能であるが，様々な実験を行うことを念頭に別プロセスとしている．

4.5 Comm. P. Analyzer

アプリケーションが利用している通信プロトコルは，安全な通信を行うために通常これら通信プロトコルは SSL とともに使われる．SSL によりデータが暗号化されるため，SSL よりも上位層となるこれら通信プロトコルの挙動はシステムソフトウェアから解析することは出来ない．アプリケーションが動的リンクライブラリを使っている場合，SSL ライブラリ関数をフックして暗号化前あるいは復号化後のデータを読むことによりプロトコル解析が可能となる．図 4 において Comm. P. Analyzer はこのためのモジュールである．このモジュールにより通信プロトコルの規約に沿って通信が行われているか解析することが出来る．

プロトコル規約に沿った通信が行われているかを示すイベントは MQTT Broker に送られる．プロトコル規約違反が発見された時には通信の遮断など緊急対処を Enforcement コンポーネントに依頼する．これらの通信は，MQTT Broker 経由で行われる．

なお，本モジュールは通信プロトコルごとに実装する必要がある．

4.6 Coordinator モジュール

図 4 に示す通り，ZT-IoT システムソフトウェアでは，MQTT Broker が IoT 機器内で Coordinator モジュールとして動作する．IoT 機器内で動作している MQTT Broker と Cloud 上で動作している MQTT Broker が連携し，機器内で動作している Monitoring および Enforcement コンポーネント，インターネット上の Cloud に配備されている Security Policy Controller コンポーネント，の 3 つで実現されているモジュール群が送受信するデータメッセージや制御メッセージの仲介を行う．MQTT によるモジュール間のデータ通信および制御通信を採用したことにより，モジュールの追加・変更が容易となる．

先に述べたとおり MQTT は topic ベースの pub/sub 通信モデルを実現している通信ライブラリの 1 つである．図 7 に，Audit，Statistics Monitor，Comm. P. Analyzer が送信する監視データ，Filter プロセス (Filter-th1 から Filter-th4 のスレッドが実行していると仮定) がそれらを受信してフィルタリングした後に監視データを送信する時の例を示す．

Filter プロセスが送信するメッセージは図 3 で例示した Pub-1 送信者と同じである．すなわち，topic 名の第 1 階層に ZT-IoT を指定し，第 2 階層は機器名を指定している．これは Cloud 上の Security Policy Controller に送信することを意図している．Security Policy Controller は，図 3

/usr/include/libaudit.h で定義されている．

```
Audit             publish topic: "local/aduit/syscall/pid1234" data: "...."
                  publish topic: "local/aduit/syscall/pid4321" data: "...."
Static Monitor    publish topic: "local/net/traffic"   data: "..."
Comm. P. Analyzer publish topic: "local/proto/dds"     data: "..."
Filter-th1        subsribe topic: "local/audit/#"
Filter-th2        subsribe topic: "local/net/traffic"
Filter-th3        subsribe topic: "local/proto/dds"
Filter-th4        publish topic: "ZT-IoT/m1/audit/syscall/pid1234" data: "..."
                  publish topic: "ZT-IoT/m1/audit/syscall/pid4321" data: "..."
                  publish topic: "ZT-IoT/m1/proto/dds" data:"..."
                  publish topic: "ZT-IoT/m1/net/traffic" data:"..."
```

図 7 MQTT 通信例

で例示した Sub-1，Sub-2，Sub-3 のように topic を購読することにより，機器 m1 から送信される audit，proto/dds，net/traffic に関するメッセージを受信することが出来る．ただし，topic 名が ZT-IoT/#のメッセージは IoT 機器の Coordinator からクラウド上の Coordinator に publish し，クラウド上の Security Policy Controller はクラウド上の Coordinator から subscribe するようにする．これは，外部から IoT 機器に対して subscribe しないことでセキュリティリスクを軽減するとともに，NAT の内側にある IoT 機器への接続など非対称ネットワークでの利用を可能にするためである．

Audit, Static Monitor, Comm. P. Analyzer が送信するメッセージの topic 名の第 1 階層は local としている．topic 名の第 1 階層に local を指定することによりは機器内で使用されるメッセージとして扱っている．すなわち，Filter プロセスの 3 つのスレッドがこれらメッセージを受信する．

5. 関連研究・システム

近年，Linux eBPF[12] を用いた様々なツールが開発されている．Tetragon[13] は，eBPF を用いてシステムコールを監視できる．Audit は監査機能だけだが Tetragon はポリシーに基づくエンフォースメントやフィルタリングも可能である．Pixie[14] も eBPF を用いて CPU プロファイルやデバッグ，ネットワークプロトコルを監視できる．SSL に対しては SSL ライブラリが呼び出される前のデータを監視する．なお，本論文執筆段階で Arm はサポートされていない．

Linux 上でシステムコールを監視する手法は Audit の他にも複数提案されている．Linux Security Module (LSM)[15] のフック関数を用いてシステムコールを監視し，マルウェア解析に用いる手法が提案されている [16]．VSyscall[17] は，プログラムの振る舞いを監視するために VMM (Virtual Machine Monitor) を用いてシステムコールの監視を行う．Qemu と KVM で VSyscall のプロトタイプを実装しており，Qemu 版では INT 80h 命令を intercept し，KVM 版では Intel VT support のプラットフォームで GUEST_SYSENTER_EIP を上書きしてページフォルトを起こさせることでシステムコールを検出する．コンテナ上のアプリケーションのシステムコールシーケンスを監視して，異常検知する手法も提案されている [18]．このシステムでは，Linux の strace ツールを用いて Docker コンテナとホストカーネル間のシステムコールをホストカーネル上で監視する．ファームウェアで監視する方法では，Ninja[19] がある．Ninja は，ARM のハードウェアサポート隔離実行環境である TrustZone を用いて，透過的にアプリケーションをトレースするシステムである．ARM Trusted Firmware で実装されており，PMU (Performance Monitoring Unit) と ETM (Embedded Trace Macrocell) を用いてシステムコールを監視する．

これら既存の監視機構は Pixie を除くと Linux カーネルシステムコールの監視である．我々が現在想定している Linux Audit はシステムコールの監視だけでなく，sshd や openssl と連携してログインや認証関係の監査データも取得できる．現在 Linux Audit を利用することで設計・実装を進めているが，これら既存の監視機構でも，我々が提案する MQTT topic 命名規則に準じた MQTT 通信送信機能を実装すれば，我々のシステムと統合できる．

オープンソースの Fluentd[20] は，様々なログファイルを集約し，解析，保存するスケーラビリティと拡張性に優れたツールである．Fluentd を使ったログ解析ツールや MongoDB や Amazon S3 に保存する機能がある．Plugin 機能を使ってデータのフィルタリングも可能である．データ集約の観点からは Fluentd と我々が検討しているシステムは実現手段の違いにある．我々のシステムは，MQTT topic 命名規則を使って体系的に監視情報を分類し選択できる枠組みを提供している．また，我々のシステムは MQTT Broker を介してセキュリティポリシに基づく Enforcement モジュールへの指示を伝達する目的にも使われるべく設計を進めている．

6. 終わりに

本論文では，我々が開発を進めているゼロトラスト環境における安全な IoT システムを実現する ZT-IoT システムソフトウェアのうち，主にセキュリティのための監視機構

実現方法に関する検討を行った．Linux カーネルが搭載されている IoT 機器上において，Audit，Statistic monitor，Filter の 3 プロセスと，アプリケーションプログラム中に動的にリンクされた通信プロトコル解析コードにより機器内の監視機構が実現される．これらアクティビティを協調動作させるために MQTT Broker を導入した．

MQTT が提供する topic 名は，Unix のファイルパス名のように”/” をつけて階層化出来る．topic 名の階層化仕様を利用し，IoT 機器内のプロセス間で交換される通信データ種類の識別，IoT 機器と Cloud 上のセキュリティポリシを決めるモジュールとの通信識別に用いた．これにより，新しい監視対象の導入や監視プロセスの置換などが容易におこなえる．今後プロトタイプ実装を進め，提案システムのオーバヘッドを測定しシステムの最適化を行う．

謝辞 本研究は，JST，CREST，JPMJCR21M3 の支援を受けたものである．

参考文献

[1] 竹房あつ子，五十嵐淳，関山太朗，松井俊浩，小野泰司，福田健介，蓮尾一郎，合田憲人，石川 裕：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，研究報告システムソフトウェアとオペレーティング・システム（OS），Vol. 2022-OS-154, No. 3, 情報処理学会，pp. 1–16 (2022).
[2] Rose, S., Borchert, O., Mitchell, S. and Connelly, S.: Zero Trust Architecture, NIST SP 800-207, Technical report, NIST (2020).
[3] OASIS: MQTT Version 5 (2019).
[4] 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).
[5] OMG: OMG Data Distribution Service (DDS) Version 1.4 (2015).
[6] Shelby, Z., Hartke, K. and Bormann, C.: The Constrained Application Protocol (CoAP), *RFC 7252*, IETF, (online), DOI: 10.17487/RFC7252 (2014).
[7] ISO/IEC: Information technology — Advanced Message Queuing Protocol (AMQP) v1.0 specification, *ISO/IEC 19464:2014* (2014).
[8] Apache: Apache Kafka, `https://kafka.apache.org/`.
[9] W3C: SOAP Version 1.2 Part 1: Messaging Framework (Second Edition) (2007).
[10] P. Saint-Andre, E.: Extensible Messaging and Presence Protocol (XMPP): Core, *RFC 3920*, IETF (2004).
[11] National Vulnerability Database: CVE-2021-28166, `https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-28166` (2021).
[12] The Linux Foundation: eBPF, `https://ebpf.io/`.
[13] Cilium: Tetragon, `https://github.com/cilium/tetragon`.
[14] PIXIE labs: How Pixie uses eBPF, `https://docs.pixielabs.ai/about-pixie/pixie-ebpf/`.
[15] Wright, C., Cowan, C., Smalley, S., Morris, J. and Kroah-Hartman, G.: Linux Security Modules: General Security Support for the Linux Kernel, *11th USENIX Security Symposium (USENIX Security 02)*, San Francisco, CA, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/11th-usenix-security-symposium/linux-security-modules-general-security-support-linux⟩ (2002).
[16] Isohara, T., Takemori, K., Miyake, Y., Qu, N. and Perrig, A.: LSM-Based Secure System Monitoring Using Kernel Protection Schemes, *2010 International Conference on Availability, Reliability and Security*, pp. 591–596 (online), DOI: 10.1109/ARES.2010.48 (2010).
[17] Li, B., Li, J., Wo, T., Hu, C. and Zhong, L.: A VMM-Based System Call Interposition Framework for Program Monitoring, *2010 IEEE 16th International Conference on Parallel and Distributed Systems*, pp. 706–711 (online), DOI: 10.1109/ICPADS.2010.53 (2010).
[18] Abed, A. S., Clancy, T. C. and Levy, D. S.: Applying Bag of System Calls for Anomalous Behavior Detection of Applications in Linux Containers, *2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–5 (online), DOI: 10.1109/GLOCOMW.2015.7414047 (2015).
[19] Ning, Z. and Zhang, F.: Ninja: Towards Transparent Tracing and Debugging on ARM, *26th USENIX Security Symposium (USENIX Security 17)*, Vancouver, BC, USENIX Association, pp. 33–49 (online), available from ⟨https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/ning⟩ (2017).
[20] Cloud Native Foundation: Fluentd V1.0 Documentation, `https://docs.fluentd.org/`.