情報処理学会論文誌 Vol.63 No.3 861–878 (Mar. 2022)

リアルタイム挙動に基づく動的アクセス制御を効率的に実現するゼロトラストアーキテクチャ

川口 信隆[1,a)] 西嶋 克哉[1] 重本 倫宏[1]

受付日 2021年5月31日, 採録日 2021年12月3日

概要：本稿では Single Packet Authorization（SPA）を拡張し，ネットワーク内で発生する TCP 通信に対してリアルタイム性が高い動的アクセス認可を効率的に実施するアーキテクチャ BiZA：Behavior based Zero Trust Architecture を提案する．BiZA は不可逆圧縮技術を活用することでリスト型情報を含む 16 種類の挙動情報を 1 つの UDP パケットに格納する．また，送信間隔を調整することで認可に係るトラヒックの増大を抑制する．これにより，従来技術と比べて多様・柔軟なアクセス制御を効率的に実施できる．評価実験を通じ，遅延 1〜10 秒以内の動的情報を活用することで高速拡散型マルウエアを被害拡大前に遮断できること，事前対策や脆弱性対策，内部犯行対策にも活用できることを検証した．また，認可処理に係るネットワークトラヒック負荷の増大は 0.2〜0.3%と軽微であることを確認した．

キーワード：ゼロトラスト，SPA，マルウエア

BiZA: Behaivor Based Zero Trust Architecture for Effective Realtime Dynamic Access Control

Nobutaka Kawaguchi[1,a)] Katsuya Nishijima[1] Norihiro Shigemoto[1]

Received: May 31, 2021, Accepted: December 3, 2021

Abstract: This paper proposes a zero trust architecture, named BiZA, that extends Single Packet Authorization (SPA) and imposes real-time dynamics access authorization on all TCP connections in networks. BiZA employs an irreversible compression technique to put 16 types of behavior information including list types (e.g., process lists) into a UDP single packet. Also, it controls the transmission intervals of the packets, making it possible to limit the traffic overhead. Thanks to the schemes above, BiZA archives more flexible and variable access control in a more efficient way compared to existing techniques. Through evaluation experiments, we confirmed BiZA can contain the spread of high-speed propagation malware by utilizing dynamic information with time delay of 1-10 sec., and it also copes with pre-incident measures, vulnerability measures and insider threats. We found that the increase of traffic overhead due to the authorization process is just 0.2〜0.3%, which is within an acceptable level.

Keywords: zero trust, SPA, malware

1. はじめに

近年，サイバー攻撃の激化やクラウド利用の普及，遠隔からのリモートアクセスの機会の増大などにともないゼロトラストの考え方が注目されている．ゼロトラストでは，原則的に自分以外の計算機を信用せず，アクセスのたびに厳格な認可を行うことが求められる．特に NIST SP800-207 [1] では，ゼロトラスト環境では計算機間のすべての通信に対して，挙動などの動的情報を利用した認可を行うことを求めている．しかし既存のゼロトラストアーキテクチャの多くでは，リアルタイム性が高い挙動情報を効率的に扱う手段が確立されておらず，保護対象とするネットワーク内の計算機間のすべての通信に動的認可を適用するのが困難であった．

本稿では Single Packet Authorization（SPA）と呼ばれ

[1] 株式会社日立製作所
Hitachi Ltd., Chiyoda, Tokyo 101–0062, Japan
[a)] nobutaka.kawaguchi.ue@hitachi.com

るゼロトラスト技術を拡張し端末やユーザの 16 種類の挙動情報を 1 つの UDP パケットに格納することで，保護対象ネットワーク内で発生する通信に対するアクセス認可を効率的に実現するアーキテクチャ Behavior based Zero Trust Architecture（BiZA）を提案する．BiZA はプロセスリストなどのリスト型挙動情報を非可逆圧縮することで多様な挙動情報を効率的に扱いアクセス制御に活用する．また，認可処理に必要なパケット送出回数を効率的に減らすことでネットワークや計算機にかかる負荷を軽減する．

評価実験を通じ，BiZA は遅延 1〜10 秒のリアルタイム性が高い挙動情報を基に WannaCry など高速拡散型マルウエアによる被害拡大を防止できるほか，Indicator Of Compromise（IOC）と連携した事前対策や脆弱性対策，内部犯行対策に活用できることを検証した．また，認可処理がネットワークトラヒックに与える影響は 0.2〜0.3%と軽微であることを確認した．

以下，2 章で本研究の背景を述べ，3 章で BiZA の詳細を説明する．4 章，5 章では評価実験結果および考察を述べ，6 章を本稿のまとめとする．

2. 背景

本章ではゼロトラストアーキテクチャの概要と BiZA の基盤である Single Packet Authorization（SPA）に関して述べる．

2.1 ゼロトラストアーキテクチャ

本節では NIST SP800-207 [1] を基に，ゼロトラストアーキテクチャの概要を述べる．ゼロトラストの前提となる考え方は，「攻撃者の侵入の可能性を排除できない限り，組織が保有するエンタプライズネットワークの信頼性は，その他のネットワーク（インターネットなど）と同程度でしかない」である．よって，自組織内からのアクセスであっても無条件には信頼できない．そこでゼロトラストでは「アクセス側に対し暗黙の信頼を置かず，被アクセス側の資産や機能に対するリスクを継続的に分析し防御すること」が求められる．

NIST SP800-207 ではこれをふまえてゼロトラストアーキテクチャが満たすべき以下の 7 要件を示している．

1. すべてのデータソースおよび計算サービスをリソース（保護対象）と見なす．
2. ネットワーク上の位置に依存せず全通信を保護する．
3. 個々のリソースへのアクセスに対してセッション単位で認可を行う．
4. アクセス可否は，アクセス要求側のアイデンティティやアプリケーション/サービスおよび資産情報に加え，挙動や環境に関する属性も含む動的ポリシに基づき決定される．
5. 組織が保有するデバイスの完全性・セキュリティ状態を監視・計測する．
6. すべての認証・認可は，アクセスの確立に先立って動的かつ厳密に行われる．
7. 組織内の資産，ネットワークインフラ・通信に関する情報を可能な限り収集しセキュリティ向上に活用する．

図 1 にゼロトラストアーキテクチャの構成を示す．主要な構成要素は Subject，Resource，Policy Enforcement Point（PEP），Policy Decision Point（PDP）である．本稿の定義では Subject は端末およびその使用ユーザを包含する．Resource は Subject がアクセスする対象であり様々なネットワークサービスやデータベースなどを含む．Subject にはいわゆる Agent が導入されており，Subject の状態を外部に発信する機能を有する場合もある．PDP は Subject から Resource へのアクセス可否を決定する機能である．PEP は Subject から Resource への通信路上に設置され PDP の決定に従い通信の許可・遮断を行う．

本稿では上記 7 要件に基づき，「セッション単位での動的認可」をゼロトラストアーキテクチャの技術的特徴の 1 つとして位置づける．そして動的認可に資する情報の中でも特に「端末やユーザなど Subject の直近の挙動情報」に着目し，リアルタイム性が高い挙動情報に基づくセッション単位動的認可を効率的に実現するアーキテクチャを提案する．

既存のゼロトラストアーキテクチャの多くは IP アドレス，ロケーション，日時・時間，端末の識別子，OS や Web ブラウザのバージョン，OS パッチ適用状況，アンチウイルスソフト・端末 firewall の ON/OFF，検知アラートなどを基にアクセス制御を行う機能を有する [2], [3], [4], [5].

一方で，既存のアーキテクチャの中には，クラウドや組織拠点に設置した PDP 上で集中的にアクセス認可判定を行うものもある．こうしたアーキテクチャ上でネットワーク内のあらゆる通信セッションに対してそのつど動的認可を行おうとする場合，認可に係るトラヒックの増大，認可に要する時間の長大化，PDP への高い負荷が生じうる，という問題がある．実際，現行のアーキテクチャの多くでは，動的認可を行うタイミングはアプリケーションや PDP へのログイン時などに限られる．さらに Subject から取得する挙動情報のリアルタイム性は 5 分のオーダであり [5]，マルウエアや攻撃者の端末への侵入で生じる状況の変化（今まで起動していなかったプロセスが実行される，Firewall などのサービスがシャットダウンされる）の把握・追随が難しい恐れがある．

ゼロトラストに資するアクセス制御の関連研究としては文献 [6], [7], [8], [9] などがある．Poise [6] は P4 Programmable Switch を用いて端末から発信された認可パケットに対する高速な認可処理を SDN 上で実現する．使用する認可情報としては，位置情報，時間，ユーザの端末操作状況などがある．eZtrust [7] は Micro Service を対象

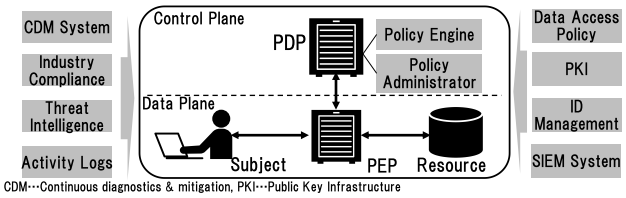


図 1 ゼロトラストアーキテクチャの構成（文献 [1] に基づく）
Fig. 1 Components of zero trust architecture (based on [1]).

に，OpenSSL のバージョン情報などを用いたパケット単位での動的認可を行う．これらの研究は認可速度や認可粒度の点で優れている一方で，認可に使用可能な情報の多様性・柔軟性が低い，認可パケットの暗号化・検証を十分に行っていない，活用場所に制約があるといった課題がある．PivotWall [8] は機密ファイルの端末間伝播の追跡に特化した手法である．TAC [9] は TCP ヘッダ中に 32 bit の識別子トークンを埋め込み PEP 上で認可する．この方式では挙動情報を用いた動的認可はサポートされていない．

2.2 Single Packet Authorization

Single Packet Authorization（SPA）は元々はシステムの firewall を動的制御する仕組みとして考案されたプロトコルである [10], [11]．仕組みの特徴は，サービスの存在を外部から隠ぺいし，SPA パケットによる事前認可を通過した Subject のみパケットの到達を許可することにある．たとえば，インターネット上からアクセスされる SSH サービスがあるとする．外部の攻撃者がスキャンなどを通じて SSH ポートの開放を発見した場合，brute-force 攻撃や脆弱性悪用による侵害リスクが生じる．SPA の仕組を使用すると，あらかじめ認可された Subject から以外のパケットは SSH サービスに到達しないため，攻撃者にサービスの存在を隠ぺいでき高い安全性を確保できる．

Cloud Security Alliance（CSA）が推進する Software Defined Perimeter（SDP）に基づくゼロトラストアーキテクチャでは，SPA を用いてクライアント–ゲートウエイ間のアクセスをセキュア化することで，Authenticate-before Connect（接続前認証）を実現する [12], [13]．認可されない限りゲートウエイの背後にあるサービスにクライアントからのパケットは到達しないため，DoS 攻撃に対する耐性が高い．また，サービスがゼロデイ脆弱性を有している場合のリスクを軽減することができる．

図 2 に SPA を用いた認可の仕組みを示す．Subject が Resource への通信開始要求を行うタイミングで SPA パケットが PEP に対して送信される．SPA パケットは UDP パケットの一種であり，ペイロードには Subject の認証情報（パスワードなど）やアクセス先の Resource に関する情報が記載される．PEP では SPA パケットを検証し通信を許可する場合に限り一時的に（たとえば 30 秒間）ポートを開放する．

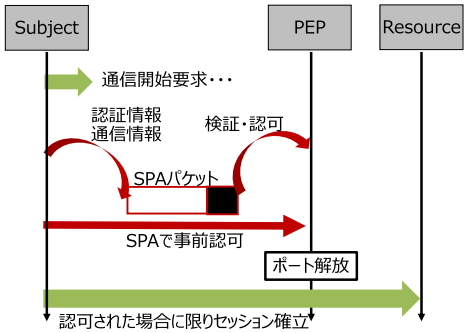


図 2 SPA における認可の仕組み
Fig. 2 Authorization mechanism in SPA.

SPA パケットの生成・送信は Agent が行う．リファレンス実装としては fwknop [14] があるが，Subject の通信開始要求に合わせて，SPA パケットを自動的に発出する機能は有しておらず，手動のコマンド入力が必要である．また SPA パケットのフォーマットは定まっておらず OpenSPA [15] などいくつかの提案がされているが，CSA では以下の 3 要件を満たすことを求めている [16]．

1. SPA パケットは，暗号化し，認証機能を持たなければならない．
2. SPA パケットは，必要な情報をすべて 1 つのパケットの中に含めなければならない．
3. SPA パケットを受け取ったサーバは，応答しないし，何も送信しない．

SPA は Authenticate-before Connect により，前節で述べた「セッション単位での動的認可」のうち「セッション単位での認可」を実現する．一方 SPA パケットの中に入れる認可情報については規定がなく，これまでの研究ではパスワードなどに限定されている．SPA を用いる SDP ベースのアーキテクチャでも認可情報はコントローラ上で中央管理する想定であり動的認可を行うタイミングはコントローラへの認証時に限られる [5]．BiZA は SPA のコンセプトに従いながら Subject の挙動情報を 1 つの UDP パケットに格納することで「セッション単位での動的認可」を効率的に実現する．

3. BiZA：Behavior based Zero Trust Architecture

BiZA の目的は，端末やユーザの直近の挙動情報を SPA を応用した認可要求パケットに格納して Subject から PEP に送信することで，保護対象ネットワーク内で発生する個々の TCP セッションに対する動的認可を効率的に実現することである．BiZA を適用する保護対象ネットワークは以下の要件を満たすものとする．

1. ペイロードサイズ 1,232 Byte までの SPA パケット（OpenSPA での推奨値）が経路上で分割されずに Subject から Resource まで配送される．

2. 任意の Subject から任意の Resource に送信される UDP/TCP パケットが共通して必ず通過する通信経路があり，その上に PEP を設置できる．
3. Subject から Resource への通信遅延の変動はたかだか 100 msec 以下であり大きな揺らぎがない．

1. および 2. は通信経路上に設置した PEP 上で SPA パケットおよび対応する TCP SYN パケットを処理するために必要である．3. は後述する，SPA パケットと TCP SYN パケットの送信間隔を決定するうえで望ましい性質である．典型的な LAN/企業内ネットワークは上記の条件を備えている．

BiZA が対象とする脅威は

1. Subject に感染した WannaCry [17] や Notpetya [18] などの高速拡散型マルウエアによる保護対象ネットワークへの被害拡大．
2. 悪意ある従業員による保護対象ネットワークからの情報漏洩などの内部犯行．

である．BiZA は事前対策，脆弱性対策，事後対策に資する動的情報を基にこれら脅威に対するリスク判定を行いアクセス制御を実施する．BiZA は動的認可に必要な挙動情報を 1 つの認可要求パケットに格納する．PEP は他のデータベースや PDP に問合せをせずに認可判定を行えるため認可に係る処理負荷が軽減される．以下に BiZA の要件やアーキテクチャ，挙動監視，アクセス制御の仕組みについて詳説する．

3.1 BiZA の要件

本節では，BiZA が満たすべき 4 条件を示す．

要件 1：保護対象ネットワーク内の Subject-Resource 間のすべての TCP セッションを認可対象とする

2.1 節で述べたとおり，ゼロトラストアーキテクチャではアプリケーションプロトコルによらず，Subject-Resource 間のすべてのセッションを認可対象とすることが望ましい．既存技術では OAuth や SSO など Web ベースの ID Federation 技術を活用して認可を実現することが多い [19]．一方 BiZA では SPA の仕組を活用し TCP セッション全般を対象にすることで，HTTP や HTTPS に加えて SSH，SMB，FTP，SMTP，POP，Delivery Optimization（P2P 型 Windows update），TELNET，UPNP，WSDAPI あるいは独自プロトコルなど，エンタプライズネットワーク内で発生する様々な TCP 通信を，アプリケーションレイヤに依存せず包括的にカバーする．また Subject-Resource の間でのプロトコル変換を必要とせず，保護対象のプロトコルに認証・認可機能がない場合でも動的認可を実現できる．BiZA は fwknop [14] と同様に Subject-Resource 間の TCP セッションを ⟨srcip, dstip, dstport⟩ の 3 タプルで識別し，認可およびセッションレベルでのアクセス制御を行う．

本稿で報告する BiZA では UDP は認可対象としないが，今後 QUIC/HTTP3 [20] に代表される UDP ベースの通信が普及することも予想される．このため UDP への対応は今後の研究課題とする．

要件 2：リアルタイム性が高い挙動情報を Subject から収集して，PEP 上で統合的な認可判定を実施する

ここでのリアルタイム性とは，後述の理由から，Subject 内で発生した過去 30 秒以内の挙動情報を認可に活用できることと定義する．たとえば，WannaCry など高速拡散型の攻撃では，感染から 40 秒で他端末への Lateral Movement が行われる．また，新たに発見された脆弱性がすぐに攻撃に悪用されることも考えられるため，対策は早急に行う必要がある．このため，認可に活用する挙動情報はできる限り最新であることが求められる．上記に示した高速拡散型マルウエアの速度を勘案すると，端末状態の更新を 5 分間隔で実施する [5] 既存技術では対応が明らかに間に合わない．このため，Agent の処理遅延なども考慮すると，挙動情報収集の望ましいリアルタイム性は，既存技術の 10 倍にあたる 30 秒以内と考える．

挙動情報の収集・送信は端末内の Agent が行う．認可に活用する挙動情報のタイプは以下の 5 種類である．PEP ではこれらを統合して認可判定を行う．

(1) サイバー攻撃に起因する可能性がある不審挙動
(2) 攻撃に対する脆弱性を有するプログラムの実行有無
(3) 期待されているセキュリティ対策の実施状況
(4) 攻撃に悪用される恐れが高いプログラムの実行
(5) 内部犯行に資する活動

(1) については，先行研究 [21] の中で，標的型攻撃者やマルウエアの侵入にともない，端末が過去に行ったことがないプロセスの起動や，これまでに通信したことがない組織内端末との通信などが発生することが示されている．そして，端末の通常活動からの逸脱度合に基づいて，攻撃者が侵入している可能性がある「不審端末」を発見できることが述べられている．BiZA では先行研究 [21] を援用し，「不審端末」の観点から Subject のリスクを評価して認可判定に反映させることで，アンチウイルスや EDR（Endpoint Detection & Response）で見逃された脅威に対応できる．不審挙動を示す Subject が必ずしも「悪性」とは限らない．しかし，不審挙動の発生頻度・パターンとアクセス先 Resource の重要度を鑑みリスクを判断することで，攻撃被害と通常業務への副作用を最小化するアクセスポリシを構築できると考える．

(2) については，TCP セッション開始要求の時点で，脆弱性を有することが明らかなプログラムの端末上での実行状態を検証する．起動中のプログラムを確認することで，OS のインベントリ機能などで捕捉しにくいポータブルアプリケーションの脆弱性にも対応する．これにより脅威が侵入するリスクがある Subject へのアクセス制限が実現できる．

(3) については，TCP セッション開始要求の時点で，端末内で本来は動作していることが期待される対策の実行有無を検証する．対象となる対策としてはアンチウイルス，ファイアウォールなどのセキュリティ対策機能や，業務監視ソフトウエアなどが考えられる．期待される対策が実施されていない場合，ユーザのポリシ違反や攻撃者による改変などが発生していると考えられ，セキュリティリスクが高い状態といえる．

(4) については，TCP セッション開始要求の時点で，OSINT などの分析 [22] で得られた最新の IoC（Indicator of Compromise）に合致する悪性プロセスが端末内で動作していないかどうかを検証する．また，本来は正規業務のために作成されたが攻撃に転用される恐れがあるツール [23] の起動状況も認可判定に用いる．

(5) については，文献 [24] などを参考に，情報漏洩など内部犯行で悪用されやすい挙動を監視対象とする．

要件 3：Agent は挙動情報の収集・送信のみを行う

BiZA では，Agent はあらかじめ決められた挙動情報の収集・送信のみを行い，運用中に外部から与えられた情報を利用した処理は行わない．たとえば，要件 2 の (2)–(4) で述べた挙動の捕捉には，Subject からの挙動情報に対して脆弱性情報や IoC といった Cyber Threat Intelligence（CTI）との突合せを行う必要がある．BiZA では，「アクセスポリシスクリプト」の形で CTI を記述して PEP 上で突合せを行う．そのため Agent に対する CTI の配信は行わない．これにより，新たな脅威や脆弱性が発生した場合に一定数以上の端末に短期間で情報配信を行うコストを抑えることができる．また，Firewall や NAT 配下にあり Push 型配信が難しい端末に対しても PEP を通じたアクセス制御ができる．

要件 4：挙動情報は self-contained な形で 1 つのパケットに格納する

SPA と同様に，Agent は要件 2 に示したすべての挙動情報，および暗号化や認証に必要な情報を 1 つの認可要求パケットに格納する．認可に必要なすべての挙動情報が 1 つのパケットに格納されているため，PEP がデータベースや PDP などに問合せを行う必要がなくなる．これにより認可処理にかかる時間を短縮することができ，問合せにかかるネットワーク負荷や PDP の処理負荷を軽減できる．また PDP がダウンすることで生じる Single Point of Failure を防ぐことができる．さらに，PEP-Subject 間で認可に関するステータスを同期する必要がないため，PEP や Subject が一時的にダウンしたり再起動したりした場合でも，その後の認可処理を円滑に再開できる．

挙動情報を 1 つのパケットに格納することの別の利点は，PEP の実装形態の幅が広がることである．本稿で示す実装例は Web Proxy やサーバで動作するプログラムであるが，ネットワークスイッチ上での実現も可能であり様々な形態のネットワークに対応することができる．

一方で上記要件 1〜4 を実現するには以下の技術課題を解決する必要がある．

技術課題 1：多様な挙動情報を 1 つの IP パケットに格納する方法（要件 2・3・4 に関連）

技術課題 2：認可要求パケットと TCP SYN パケットの発信タイミングの調整（要件 1 に関連）

技術課題 1 では，端末上で実行されているプロセスのリストなど，比較的大きなデータサイズを持つ情報の扱いが課題となる．BiZA では Bloom Filter [25] という非可逆情報圧縮技術を使うことでリスト型情報のデータ量を削減し，この問題を解決する．また，リスト型情報を扱えることで PEP 上で CTI との突合せができる．

技術課題 2 では，認可要求パケットを送信してからどの程度の時間間隔で TCP SYN パケットを発信するかが課題になる．間隔が短すぎると認可処理が完了する前に SYN パケットが PEP に届き破棄される恐れがある．一方，間隔が長すぎると通信効率が低下する．特に Web 通信など短時間に一定数以上の TCP コネクションが発生しうるプロトコルでは，この問題が深刻になる．BiZA は PEP に認可要求パケットを最後に送信してからの時間間隔を基に TCP SYN パケットの発信タイミングを決めることで，この問題を解決する．

なお，SPA パケットを受信した PEP が応答を返すことで，TCPSYN パケットの送信タイミングを Agent が決定することも可能であり，通信確立の効率性の点ではこちらの方が優れているともいえる．しかし BiZA では以下 2 点の理由により応答パケットを用いない設計を採用する．

1 つ目の理由は CSA が定める要件 3 に関連する．この要件の設定理由は，ネットワークを “black” 化することで，攻撃者に対して攻撃先の狙いを定める情報をできる限り与えないことである [13]．たとえば，何らかの手段で自身の端末をネットワークに接続した攻撃者が，ネットワーク中に SPA パケットを送信し，その応答の有無や応答内容を基に PEP や Resource の場所や IP アドレスを把握するという偵察活動を事前に行う可能性がある．

2 つ目の理由は PEP の負荷を減らすことで，より多くの端末に対応するためである．応答パケットには暗号化処理が必要なため PEP に一定の負荷がかかる．また，応答を返すことで BiZA に関連してネットワークを流れるパケット数は 2 倍になる．

ただし一方で，SPA の実装形態の 1 つである OpenSPA ではプロトコル設計上，応答パケットをサポートしている．このため，応答パケットを返すことが完全に否定されているわけではなく，通信確立の効率性，通信負荷，セキュリティのどれを重視するかで設計方針が変わるといえる．そこで本稿では応答パケットを返すタイプの設計・実装も別途実施し，通信確立にかかる時間および PEP の処理性能

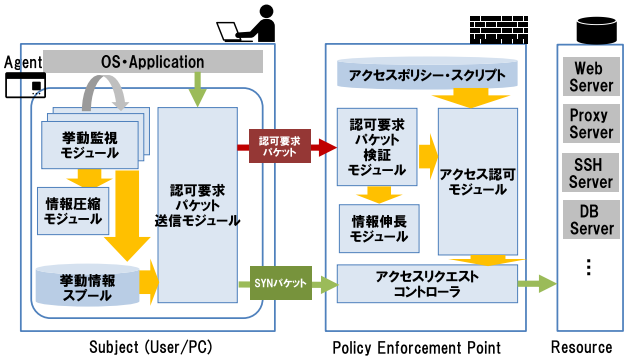


図 3 BiZA の構成
Fig. 3 Components of BiZA.

への影響を比較評価する．

3.2 アーキテクチャ

図 3 に BiZA のアーキテクチャ概要を示す．BiZA は SPA 同様に Subject，Resource および PEP から構成される．BiZA はアクセス認可において PEP-PDP 間のやりとりを極力減らすことで高いリアルタイム性が求められる挙動情報を用いた認可処理の効率化を図る．

一方，比較的変動が少なく秒単位のリアルタイム性が必要でないと考えられる情報（後述するアクセスポリシスクリプトや，Resource の機密レベル，ユーザの役職など）については PDP 上で管理し定期的に PEP に配信する，または PEP から PDP に問い合わせた結果を一定期間キャッシュするといった利用を想定する．本稿では挙動情報を活用したセッション単位の動的認可の効率化の説明に重点を置き，PEP-PDP 間の定期的な情報配信の詳細には言及しない．

Subject には Agent が導入されており，要件 3 に従い端末やユーザの挙動に基づきアクセス認可要求を PEP に対して行う．Agent の主要構成要素は，挙動監視モジュール，情報圧縮モジュール，認可要求パケット送信モジュール，挙動情報スプールである．

挙動監視モジュールは要件 2 で述べた観点から端末・ユーザの挙動を監視し，挙動情報として出力する．出力される挙動情報の型は「数値型」および「リスト型」に大別される．挙動監視モジュールが挙動情報スプールを更新する時間間隔のパラメータを P_{update} とする．

ここで，技術課題 1 でも述べたとおり「プロセスの一覧」「サービスの一覧」といったリスト型の挙動情報はその特性上データサイズが比較的大きくなり，そのままの形で 1 つのパケットに格納するのは難しい．そこで，情報圧縮モジュールは，リスト型の挙動情報に対して Bloom Filter を適用し，データサイズを削減する．圧縮処理が完了した挙動情報は挙動情報スプールに保管される．

認可要求パケット送信モジュールは Subject が送信する Resource を宛先とするすべての TCP SYN パケット（以下，アクセス要求パケットと呼称）を捕捉する．そして，挙動情報スプールに記録された挙動情報を基に認可要求パケットを生成し PEP に送信する．また，技術課題 2 で述べたとおり，PEP 送信から適切な時間差で捕捉しておいたアクセス要求パケットを送信する．

PEP は Resource と同一の計算機上，あるいは Subject と Resource とをつなぐ通信路上に位置する．このため，PEP は Subject から Resource に向けられたアクセス要求パケットを捕捉できる．PEP は，アクセスポリシスクリプト，認可要求パケット検証モジュール，情報伸長モジュール，アクセス認可モジュール，アクセスリクエストコントローラから構成される．

アクセスポリシスクリプトは JSON で記述されたアクセス判定を規定するスクリプトである．スクリプトでは，Subject 属性および Resource 属性に加えて挙動情報を基にした認可判定ロジックを論理式で表現する．アクセスポリシスクリプトは，PDP 上で組織のセキュリティ管理者により作成され定期的に PEP に配信されることを想定する．

認可要求パケット検証モジュールは Subject からの認可要求パケットを受信して電子署名の検証を行う．その後，認可要求パケットはアクセス認可モジュールに渡される．

アクセス認可モジュールでは，まず情報伸長モジュールを用いて，圧縮された挙動情報をアクセスポリシスクリプトに記述された “キーワード”（詳細は後述）を参照しながら伸長化する．次いで，伸長化された挙動情報に対してアクセスポリシスクリプトを適用し，Subject の Resource へのアクセス可否を決定する．結果はアクセスリクエストコントローラ内に記録される．

アクセスリクエストコントローラは，Subject・Resource の IP アドレスおよび Resource のポート番号の組合せで，アクセス要求パケットを通過させる期間，あるいは遮断する期間を規定する．そして，Subject から受信したアクセス要求パケットに対してアクセス制御を適用する．

3.3 挙動監視モジュール

挙動監視モジュールは，高速拡散型のサイバー攻撃および内部犯行に関連するリスクを測るために，Subject の 16 種類の挙動を監視する．監視対象の一覧を**表 1** に示す．各項目は 3.1 節要件 2 で述べた 1 つ以上の挙動情報タイプに対応する．たとえば#1 で示すプロセスハッシュ値は，脆弱性を持つプログラムの実行，動作が期待されているプログラムの実行，リスクが高いプログラムの実行を捕捉できる．

これらの監視項目は，「特定の攻撃を検知する」というよりも「端末やユーザのリスクを定量化する」ことを目的としている．最終的に，アクセス先 Resource，Subject の属性，および端末・ユーザ挙動のリスクを総合して，通信の通過・遮断を判定するのは PEP，および PEP が参照するアクセスポリシスクリプトの役割である．

表 1 挙動監視項目一覧
Table 1 Monitored behavior.

#	監視項目名	内容	挙動情報タイプ (3.1 要件 2)					出力型
			(1)	(2)	(3)	(4)	(5)	
1	プロセスハッシュ値	監視時に起動しているプロセスの実行ファイル，および rundll32 の引数として指定された DLL の SHA1 ハッシュ値のリスト	—	✔	✔	✔	✔	リスト
2	不審プロセス	過去 window 時間内に起動されたが，それより前の時刻には実行された履歴がないプロセス，および rundll32 の引数として指定された DLL の数．先行研究[21]より援用	✔	—	—	—	✔	数値
3	アドレススキャン	端末が過去 window 時間内に TCP 接続した履歴がある端末のうち特定のアドレス空間に属するものの数．先行研究[21]より援用	✔	—	—	—	—	数値
4	ポートスキャン	端末が過去 window 時間内に TCP 接続した宛先ポート番号の種類	✔	—	—	—	—	数値
5	不審アクセス	過去 window 時間内に端末が TCP 接続した宛先端末のうち，それより前の時刻のアクセス履歴がないものの一覧（上限数を超える場合は，アクセス回数上位のみ）．先行研究[21]より援用	✔	—	—	—	—	リスト
6	ユーザアイドル時間	最後にキーボード・マウス操作が行われてからの経過時間（秒単位）．ユーザ不在時に活性化する脅威をとらえるために測定	✔	—	—	—	—	数値
7	ファイル作成数	過去 window 時間内に特定の拡張子（たとえば pdf, docx）を持つファイルが作成された回数．先行研究[21]より援用	—	—	—	—	✔	数値
8	PowerShell スクリプト実行	過去 window 時間内に起動されたが，それより前の時刻には実行された履歴がない PowerShell スクリプトのうち，Antimalware Scan Interface (AMSI)により”Warning”が出されたものの数[26]	✔	—	—	—	✔	数値
9	印刷回数	過去 window 時間内にプリンタを介して印刷を行った回数	—	—	—	—	✔	数値
10	外向きトラヒック	過去 window 時間内に発生した Web Proxy との TCP 接続のうち，最大のアウトバウンドトラヒック量．先行研究[21]より援用	✔	—	—	—	✔	数値
11	不審 USB メモリ接続	過去 window 時間内に，過去に接続履歴がない識別子の USB メモリが接続された回数	—	—	—	—	✔	数値
12	マイク無効化	情報漏洩対策として，端末のマイクが無効になっているか	—	—	✔	—	—	数値
13	実行ファイル生成	過去 window 時間内に実行ファイルが端末内で生成された回数（短時間に複数作成された場合は，まとめて 1 回の作成と見なす）．先行研究[21]より援用	✔	—	—	—	—	数値
14	カメラ無効化	情報漏洩対策として，端末のマイクが無効になっているか	—	—	✔	—	—	数値
15	起動サービス	監視時に起動しているサービスのリスト	—	✔	✔	✔	—	リスト
16	HotFix	監視時に端末に適用されている Windows HotFix 一覧	—	✔	✔	—	—	リスト

監視項目の選定は，先行研究 [21]，IPA の内部不正調査結果 [24]，MITRE ATT&CK [27]，PowerShell などを悪用した Fileless 型攻撃に対する知見 [28], [29] に基づく．

図 4 に BiZA が対象とする攻撃フェイズと監視項目との対応を示す（図中の#は表 1 に対応）．拡散型攻撃に対しては，端末とサーバの間，または端末とフォワード Web プロクシとの間に PEP を設置することで，被害拡大を防止することを想定する．内部犯行については，端末とサーバの間，端末とネットワークプリンタの間，または端末とフォワード Web プロクシとの間に PEP を設置することを想定する．端末と USB メモリの間に PEP を設置することは難しいが，サーバから収集したファイルを USB に書き出す行為を繰り返し行う攻撃者に対しては，サーバとの間の PEP でアクセス制御を行うことができると考える．

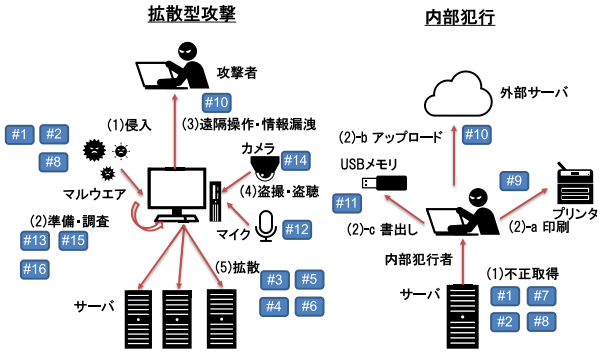


図 4 攻撃フェイズと監視項目との対応
Fig. 4 Map between attack phases and monitored behavior.

3.4 リスト型挙動情報の圧縮処理

情報圧縮モジュールは，リスト形式の挙動情報に対して圧縮処理を行うことでデータサイズの削減を実現する．ZIP などの可逆圧縮を使う方式では，乱数性が高いハッシュ値のリスト型に対して圧縮効果が非常に低い．そこで，圧縮方式として非可逆圧縮方式である Bloom Filter [25] を用いる．BiZA は表 1 の#1，#15，#16 の監視項目に対して Bloom Filter を適用する．#5 はリスト型をそのまま扱うため圧縮処理は行わない．

Bloom Filter は，あるデータが集合のメンバであるかどうかのテストに使われるデータ構造である．まず，ある固定長のビット配列を用意し，全要素を ‘0’ で初期化する．次に登録したいデータに対しいくつかのハッシュ関数を適用する．最後に，ビット配列上で，先に求めたハッシュ値に対応する添え字の要素を ‘1’ に変更する．ビット配列に特定データが登録されているか検証する際は当該データのハッシュ値に対応する要素がすべて ‘1’ であることを確認

する．

Bloom Filter は ZIP など可逆圧縮方式と比べて圧縮効率が高い．その一方，ハッシュ値の衝突が原因で，実際には未登録のデータが登録されていると誤判断される，偽陽性のリスクがある．一方原理上，偽陰性は発生しない．このため，認可判断を誤るリスクを鑑み，許容可能な偽陽性率を設定する．必要なビット配列長 L は，許容可能な偽陽性率を P_{FP}，登録したいデータ数を N とすると以下のように求められる．

$$L = -(N \times \ln P_{FP})/(\ln 2)^2 \quad (1)$$

3.5 認可要求パケットの構成

表 2 に認可要求パケットのフォーマットを示す．パケットには挙動情報のほかに，ヘッダや端末・PEP の識別子，鍵や電子署名などが含まれる．Ethernet における MTU は 1,500 Byte である．一方 OpenSPA では，IP ヘッダなどが操作される可能性を考慮すると，インターネットを通過する SPA のペイロードは 1,232 Byte 以下になることを推奨している．これを勘案すると，挙動情報を格納する#10 Context のデータサイズの最大値は状況に応じて 664〜904 Byte の間をとる．Context は各挙動情報を格納するフィールドから構成される．各フィールドは，挙動情報の識別子（2 Byte）および挙動情報のデータサイズ（2 Byte），および挙動情報本体となるペイロードから構成される．このため 1 つの挙動情報を格納するのに必要なフィールド長は最低 5 Byte となる．

フィールド#6〜#10 は 128 bit 長の AES 鍵（#12）で暗号化される．その後フィールド#1~#10 に対応する HMAC（#11）が付与される．AES 鍵（#12）は PEP の公開鍵で暗号化され，Agent 秘密鍵による署名が#13 として施される．Agent が PEP と初めて通信する場合は#12 にある AES 鍵を演算コストが高い RSA で暗号化し電子署名を施す必要がある．1 度 AES 鍵を共有できればパケットごとに RSA による暗号化や電子署名は必要なくなる．しかし，Subject・PEP の再起動やセキュリティ対策など何らかの理由で AES 鍵が定期的にリセットされた場合でも再同期できるよう全パケットに#12，#13 を付与する．なお，RSA および ECDSA の鍵長には，NIST SP800-57 [3] で 2031 年以降も利用可能な 128 bit 強度暗号に指定されている 3,072 bit，256 bit をそれぞれ用いる．

#3 にある ID Field にはリプレイ攻撃を防ぐための乱数が記録される．PEP は#5 記載の Device Id で署名検証に用いる Agent の公開鍵を特定する．#6 ユーザ識別子としては，端末にログインしているユーザアカウント名を使用する．#8 の ACCESS ではアクセス要求パケットの TCP ヘッダ記載の 4tuple を指定する．ただし，NAT 配下にある Subject がインターネット上の Resource にアクセスするとき，PEP も NAT 外にある場合は，SrcIP には NAT 外側のグローバル IP を設定する．PEP も NAT 配下にある場合は Subject に割り振られた IP アドレスでよい．

表 2 認可要求パケットのフォーマット
Table 2 Authorization request packet format.

#	Field	説明	Byte 長
1	Header	パケットが認可要求パケットであることを示す識別子	2
2	Version	パケットのバージョン	2
3	ID Field	リプレイ攻撃対策用の乱数	16
4	Timestamp	パケット生成日時	4
5	Device Id	端末識別子	16
6	User ID	ユーザ識別子	16
7	Flag	パケット種別を示す(将来の拡張に備えてリザーブ)	2
8	ACCESS	アクセス要求パケットの Src. IP, Dst. IP, Src. Port, Dst. Port	12
9	Reserved	その他用途用にリザーブ	16
10	Context	挙動情報	664~904
11	HMAC	#1~#10 の HMAC（#12 の AES 鍵を使用）	32
12	Encrypted AES Key	#6~#10 を暗号化した AES 鍵を PEP 公開鍵(RSA3072bit)で暗号化したデータ	384
13	Signature	#12 に対する Agent の電子署名（ECDSA 256bit）	64

リスト 1 認可要求パケットモジュールの疑似コード
List 1 Pseudocode of auth. request packet module.

```
1  while(true):
2     packet=captureAccessReqPacket()
3     PEP_addr = getPEP(packet.dst)
4     n=countAuthReq(PEP_addr, packet, P_req)
5     if n==0 or sendJudge(n, packet_loss):
6        sendAuthReqPacket(PEP_addr)
7     t=lastTransmission(PEP_addr, packet)
8     interval=getInterval(t, P_req, T_process)
9     sleep(interval)
10    sendAccessReqPacket(packet)
```

3.6 認可要求パケットの送信制御

認可要求パケット送信モジュールは端末からのアクセス要求パケットを捕捉し，必要に応じて認可要求パケット送信する．また適切なタイミングで，捕捉したアクセス要求パケットをネットワーク上に配信する．リスト 1 にモジュールの疑似コードを示す．

Line.3 では Line.2 で捕捉したアクセス要求パケットの送信先を基に当該 Resource へのアクセス要求に制御を行う PEP を決定する．認可要求パケットモジュールは Resource と PEP の公開鍵情報の対応表を持つことを想定する．

Line.4 では，Line.3 で決定した PEP に過去 P_{req} 時間以内に，アクセス要求パケットの〈送信先アドレス，送信先ポート番号〉の組合せに対して，認可要求パケットを送信した回数 n を求める．P_{req} には，PEP が通信をいったん

認可した場合にアクセス要求パケットを通過させる時間 P_{allow}（たとえば 10 秒）より小さい値が設定される．

Line.5 では認可要求パケットを送信するかの判定を行う．$n = 0$ であれば明らかに送信対象になる．次に，$n > 0$ の場合，ネットワークで発生しうるパケットロスを考慮して，過去に送った n 個の認可要求パケットがすべて宛先 PEP に到達しなかった確率を求め，これを基に新たに認可要求パケットを送信するかどうかを決める．すなわち，認可要求パケットを新たに送信する確率 P_{send} は，想定最大パケットロス率を P_{loss} とおくと

$$P_{send} = \mathrm{P}_{loss}^{\mathrm{n}} \tag{2}$$

で求められる．これにより短時間に多くのアクセス要求パケットが発生した場合にも，そのつど認可要求パケットを送ることなく効率的な対応が可能になる．Line.6 では，挙動情報スプールに保管された挙動情報を基に先述のフォーマットを持つ認可要求パケットを作成して PEP に送信する．

認可要求パケット送信直後にアクセス要求パケットを送信すると，認可処理が完了する前にアクセス要求パケットが PEP に届いてしまい遮断される恐れがある．そこで，認可要求パケットとアクセス要求パケット送信の時間間隔（*interval*）を調整する必要がある．Line.7 では PEP に対して最後に認可要求パケットを送信した時刻 t を求める．Line.8 では，現在時刻と t の差である *gap*，P_{req} および，PEP が認可要求パケットを受信してから処理完了までの間の十分な時間 $P_{process}$ を基に *interval* を求める．$P_{req} \leqq gap$ の場合，$interval = P_{process}$ とする．$P_{req} > gap$ の場合，現在時刻において認可要求パケットの処理が完了していない確率に応じて *interval* を設定する．ここでは $P_{process}$ に対する *gap* の比率がが高いほど処理が未完了な確率は指数関数的に減少すると仮定し，*interval* を以下の式で導出する．

$$interval = P_{process} \times e^{-\frac{gap}{P_{process}}} \tag{3}$$

$P_{process}$ の値としては，ネットワークの利用頻度などを考慮し，余裕を持った値の設定が必要になる．Windows では TCP SYN パケットの最初の再送間隔が 1 秒である．このため，T_{sign}，を Agent での認可要求パケット生成に要する時間，$T_{resource}$ を端末から Resource までの伝送遅延時間とすると，

$$P_{process} + T_{sign} + T_{resource} < 1 \tag{4}$$

が成り立つことが望ましい．本稿では，対象とするネットワークの特性に応じて，BiZA の管理者がヒューリスティックに $P_{process}$ の値を事前に設定することを想定する．ネットワークの混雑状況や Agent，PEP の処理性能などに応じて $P_{process}$ を動的に変える方法については今後の課題である．

Line.9–10 が示すように，*interval* 時間後にアクセス要求パケットは送信される．

PEP 側での認可要求パケット処理時間が *interval* よりも大きくなってしまった際は，アクセス要求パケットは PEP にドロップされる．この場合には端末がアクセス要求パケットを何回か再送する（Windows 10 の場合，1 秒後，2 秒後，4 秒後，8 秒後）．再送パケット受信までに PEP の認可処理が完了すれば，通信を確立することができる．

PEP が受信した認可要求パケットに含まれる挙動情報は最大で

$$T_{delay} = P_{update} + P_{req} + T_{sign} + T_{PEP} \tag{5}$$

だけ過去の情報である（T_{PEP} は端末・PEP 間の伝送時間とする）．$P_{req} = 0$ とすると，全アクセス要求パケットと 1 対 1 の対応で認可要求パケットが発出されるため 3.1 節要件 1 を厳密に満たす．しかし Web 通信などではほぼ同一タイミングで同一宛先に対し複数の TCP コネクションが発生することが多い．たとえば，ある端末から Web プロクシサーバへの TCP コネクションの発生時間分布を調べたところ，連続する 2 つのコネクションの発生間隔が 1 秒以下となる割合は 92%，0.1 秒以内となる割合は 67%となった．これらすべてに認可要求パケットを発行しても「リアルタイム性が高い挙動情報を用いた認可」としての効果は低い一方で，端末・PEP の処理負荷が増大する．このため，$0 < P_{req} < P_{allow}$ の間で，要求されるリアルタイム性と効率との観点から適した値を P_{req} に設定することが望ましい．

3.7 アクセスポリシリクエスト

BiZA が扱うアクセスポリシスクリプトは JSON 形式で記述され，以下の 4 項目を含む．

1. subject：端末やユーザの識別子や属性に関する条件を規定する論理式
2. resource：リソースの識別子や属性に関する条件を規定する論理式
3. context：挙動情報に関する条件を規定する論理式
4. effect：subject，resource，context がすべて満たされた場合の認可判定（「通過」= allow,「遮断」= deny）

図 5 にアクセスポリシスクリプトの一例を示す．この例では，表 1 に示した挙動監視モジュールのうち，#2，#3，#4，#6，#15 が満たすべき条件を規定している．すべての条件が満たされた場合に限り，context が満たされたことになる．#15 はリスト形式で示されるサービス一覧の中に `ABC EDR` と `XYZ Service` がともに含まれることを条件としている．前述のとおり#15 は Bloom Filter によって圧縮されるため，実際には認可要求パケットのビット配列

```
{
    "effect": "allow",
    "rules": {
        "subject": {"$.user": {"condition": "Equals", "value": "Alice"}},
        "resource": {"$.dstip": {"condition": "CIDR", "value": "10.0.0.0/8"}},
        "context": {
                  "$.mod_2": {"condition":"Lt", "value":5},
                   "$.mod_3": {"condition":"Lt", "value":20},
                   "$.mod_4": {"condition":"Lt", "value":20},
                   "$.mod_6": {"condition":"Lt", "value":120},
                   "$.mod_15":{"condition":"AllOf","values":[
                          {"condition":"AnyIn","values":["ABC EDR"]},
                          {"condition":"AnyIn","values":["XYZ Service"]}
                          ]
                      }
                  }
    },
    "priority": 100
}
```

図 5 アクセスポリシスクリプトの一例
Fig. 5 An example of access policy script.

の中に上記文字列が記録されているかを PEP 側で検証することになる．以後，本例のようにビット配列と対応付けられる，ポリシスクリプト中の文字列を「キーワード」と呼称する．今回の設計では#1，#15，#16 に対応するアクセスポリシスクリプト内の文字列が「キーワード」となる．

スクリプトの文法の詳細は割愛するが，Python ベースの Attribute based Access Control（ABAC）エンジンである py-ABAC [31] に準拠し，`AND/OR/NOT` といった柔軟な条件設定を記述できる．

スクリプトの最後にある priority は，アクセス認可モジュールが認可処理で参照する際の優先度を示す．

3.8 認可要求パケットの検証処理

PEP 内の認可要求パケット検証モジュールは，受信した認可要求パケットを復号して真正性を検証する．なお，端末から最初にパケットを受信する際は AES 鍵の復号および電子署名の検証を行い，復号済みの AES 鍵およびその電子署名をキャッシュする．以降，受信した電子署名とキャッシュを照合し，ヒットする場合には復号済みの AES 鍵をそのまま用いる．これにより計算量が多い公開鍵暗号の復号処理回数が削減され，検証処理時間の短縮化が実現できる．

次に，非可逆圧縮されたリスト型挙動情報を表現するビット配列内にアクセスポリシスクリプトに記載されているキーワードが含まれているかどうか確認する．キーワードが含まれている場合，ビット配列をキーワードに置き換える．たとえば，図 5 の例では，端末内で `ABC EDR` または `XYZ Service` というサービスが起動している場合，ビット配列は，{“`ABC EDR`”, “`XYZ Service`”} に置き換わる．

最後にパケット内の各情報を JSON 型式のリクエスト文に変換し，アクセス認可モジュールが可読可能な形にする．このとき，リクエスト文内には挙動情報に加えて，現在時刻および現在日が平日/休日かの情報を付与する．これによりアクセス認可モジュールは挙動情報 16 種類 + 時間に関する情報 2 種類の計 18 種類の context を使った認可処理を行うことができる．

3.9 アクセス認可処理

アクセス認可モジュールは認可要求パケット検証モジュールが生成したリクエスト文とアクセスポリシスクリプトを基に認可処理を行う．認可処理の結果は「通過」と「遮断」の 2 通りである．「通過」の場合，端末からリソースの当該ポート番号への TCP 接続が P_{allow} の間許可される．

「遮断」の場合，PEP の設定に応じて「完全遮断」もしくは「一時遮断」のいずれかが実施される．完全遮断の場合，一定期間（以後，P_{deny} という）の間，TCP 接続および同端末からの認可要求パケットはすべて遮断される．

一方，一時遮断では P_{deny} の遮断期間の後，P_{allow} の間は同端末から Resource へのアクセスを無条件に許可する．これにより，実際には内部犯行やマルウエアに関連していない端末がリスクが高いと誤判断された場合でも，後から通信可能な期間を設けることで復旧できる．特に，P_{deny} を TCP 接続の最大タイムアウト時間（Windows 10 では 15 秒未満）に設定すれば，「遮断」の場合でも最終的には TCP 接続を確立できる．

一時遮断は内部犯行には必ずしも有効ではないがネットワーク内外へのマルウエアの拡散活動を遅鈍化できる．このため，特に対応したい脅威が拡散型攻撃である場合，「一時遮断」の有効性は高い．

3.1 節で述べたとおり，PEP は Resource 上，もしくは Subject と Resource を結ぶ通信路上で動作していることを前提とする．これにより Subject からのアクセス要求パケットを捕捉できる．アクセスリクエストコントローラは，認可処理結果に従い補足されたアクセス要求パケット

を処理する．「認可」「遮断」どちらの判定もないパケットは default-deny の原則で破棄する．

4. 評価実験

本章では BiZA の評価について説明する．まず実装および実験条件について述べ，その後評価結果を示す．評価結果に関しては，認可要求パケットの処理性能，通信遮断性能，ユースケース・他技術との比較，の 3 つの観点で説明する．

4.1 実装

Agent および PEP の実装には Microsoft 社の .NET3.1/C# [46] および Python を用いた．アクセス要求パケットの捕捉し暗号化処理には PyDivert 0.4.1 [32]，Pycryptodomex 3.10.1 [33] をそれぞれ使用した．先述のとおりアクセスポリシスクリプトの実装は Py-ABAC 0.4.1 [31] を用いた．

4.2 実験条件

表 3 に実験で用いる Subject および PEP のスペックを示す．Resource は PEP と同一の計算機で動作する．PEP と Subject は同一ネットワーク上の異なるサブネットに設置されている．**表 4** に実験で用いる Agent，PEP のパラメータのデフォルト値を示す．window の値は，多くのマルウエアは起動から 1〜2 分で主たる挙動を発現するという知見 [34] に基づく．P_{FP} の値は監視項目によって違う．これは，プロセスハッシュ値に関する監視項目#1 は，実運用上多くの IOC とマッチングされる機会が多いという仮説に基づく．P_{loss} は，同一のエンタプライスネットワークであることを考えると十分に大きなパケットロスの想定値である．$P_{process}$ はヒューリスティックに設定した．P_{allow} は fwknop における標準設定は 30 秒だが，今回は attack surface を最小化することを重視し OpenSPA の推奨（数秒以内にすること）に従い，より短い 3 秒とした．P_{update} のデフォルト値は後述の実験結果を反映している．P_{deny} は認可結果が「遮断」であっても，再送パケットにより TCP コネクションが遅延されて確立される値を設定している．また断りがない限り PEP は図 5 に示したものと同等のアクセスポリシスクリプトを読み込んでいるものとする．また Agent は事前に Subject 上で 1 カ月動作しており，監視項目#2 などが過去の活動発生履歴の確認に用いる正常活動のプロファイルを作成済みである．

4.3 処理性能

4.3.1 認可要求パケットサイズ

図 6 に実験中に作成された認可要求パケットのデータサイズを示す．Bloom Filter による圧縮前，データサイズは 5,000 Byte を超え Ethernet の 1 フレームには収まらない．一方，圧縮後のサイズは最大で 77%減り（圧縮前 5,336 Byte, 圧縮後 1,224 Byte），Ethernet の 1 フレームでの伝送が可能なほか，文献 [15] で述べられたインターネットを経由する際の推奨サイズ 1,232 Byte を下回る．このため SPA の要件を満たすといえる．

ただし Bloom Filter のサイズは登録される要素数に比例する．たとえば非常に多くのプロセスが起動する環境では，監視項目#1 に係わるデータサイズが増大し目標とするデータサイズを上回る可能性がある．実験中，認可要求パケットのデータサイズが Ethernet の 1 フレームを超えることはなかったが，Subject の動作環境によっては P_{FP} の値を調整し圧縮率を上げる必要がある．

表 3 Subject/PEP のスペック
Table 3 Spec. of Subject/PEP.

機能	形態	リソース
Subject	実マシン	OS: Windows10 Pro, CPU: Intel Core i5-8400 2.8Ghz, Memory:16.0GB
PEP	VM	OS: Windows 10 Pro, CPU: Intel Xeon E5-2640 2.4Ghz 2 コア, Memory: 8GB

表 4 パラメータのデフォルト値
Table 4 Default values of parameters.

パラメータ	デフォルト値
window（すべての監視項目で共通）	300 sec
P_{FP} (監視項目#1)	10^{-6}
P_{FP} (監視項目#15)	10^{-4}
P_{FP} (監視項目#16)	10^{-4}
P_{loss}	0.1
P_{update}	10 sec
P_{req}	1 sec
$P_{process}$	200 msec
P_{allow}	3 sec
P_{deny}	14 sec

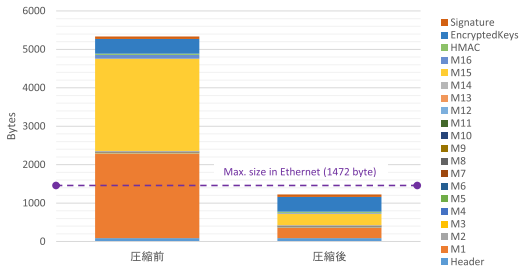


図 6 認可要求パケットのデータサイズ
Fig. 6 Data size of authentication request packet.

4.3.2 認可処理のリアルタイム性

図 7 に P_{update} と Subject の CPU 使用率との関係を示す．$P_{update} \geqq 10$ 秒では CPU 使用率は 1%台であり比較的負荷は小さいことが分かる．$P_{update} = 1$ 秒では CPU 使用率は 10%まで上昇するが，高いリアルタイム性を必要とする環境では十分実用可能な範囲と考える．またこの結果より，BiZA は要件 2 で掲げた目標値である 30 秒以内のリアルタイム性を満たすといえる．

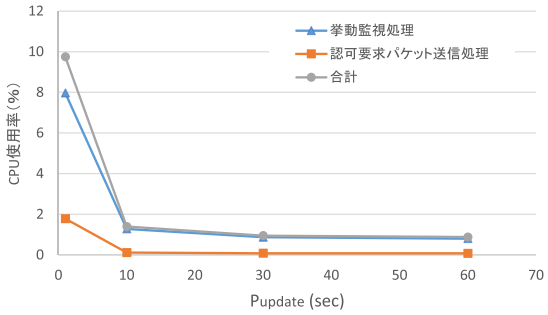


図 7 P_{update} と演算負荷
Fig. 7 Effect of P_{update} on processing time.

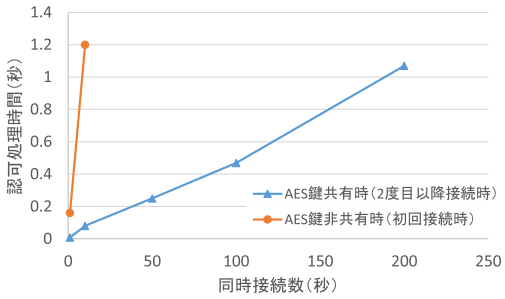


図 8 PEP における認可処理時間
Fig. 8 Authorization processing time at PEP.

4.3.3 認可処理負荷

図 8 に複数の Subject から PEP を介して Resource に同時接続を行った場合を想定した，PEP の認可処理時間を示す．処理時間は，Subject から PEP に初めて認可要求パケットを送る場合と，AES 鍵を共有済の場合とで大きく異なる．初回接続時は公開鍵を使用し，1 パケットの処理には平均 167 msec（標準偏差 9.7）を要する．一方 AES 鍵が共有済みのとき，パケット処理は 5〜6 msec の間（平均 5.4 msec）で完了する．このためすべての Subject と鍵共有が完了している場合，PEP は秒間平均 187 個の認可要求パケットを処理できる．

4.3.4 Web アクセス処理時間

次に認可要求パケットの送信制御の効果を示す．実験では Resource 上に設置した簡易 Web サーバでホストされる Web ページに Subject から google chrome を介して http アクセスを行い処理時間を測定した．ここで処理時間は，最初のアクセス要求パケットと最後のアクセス要求パケットの発出間隔の時間差である．当該 Web ページのコンテンツ取得には通常 17 回の TCP コネクションをともなう．なお Subject-PEP 間で AES 鍵は共有済みとする．

表 5 に測定結果を示す．$P_{process}$ = 200 mec のとき，BiZA がない場合と比べ処理時間は 0.25 秒増加する．このうちの多くは最初のアクセス要求パケット発出にかかる時間である．また提案方式では interval が固定の場合と比べて処理時間を 0.455 秒短縮しており，式 (3) による通信効率化の効果が示された．また $P_{process}$ = 100 mec の場合 200 mec と比べて処理時間は短くなる．しかし AES 鍵未共有時の認可処理時間が平均 167 msec かかることや，処理までの待ち時間が発生するケースを考慮すると $P_{process}$ の値は 200 msec が望ましいと考える．なお，実験中に認可失敗にともなうアクセス要求パケットの再送は起きなかった．

表 5 Web アクセス処理時間
Table 5 Web access processing time.

BiZA 無し（通常の Web アクセス）			0.252 sec
BiZA	$P_{process}$= 200mec	*interval*~Eq.(3)（提案）	0.507 sec
		interval=$P_{process}$（比較対象）	0.951 sec
	$P_{process}$= 100mec	*interval*~Eq.(3)（提案）	0.375 sec
		interval=$P_{process}$（比較対象）	0.477 sec
応答型 BiZA			0.314 sec

表 6 認可要求パケットのネットワーク負荷
Table 6 Network overload caused by auth. request packets.

秒間当たりの認可要求パケット送信数	0.076/sec
総パケット数に占める認可要求パケット数の割合	0.2%
総トラヒック量に示す認可要求パケットの割合	0.3%
認可要求パケット送信数／TCP セッション数	0.18

また，表 5 には比較対象として，SPA パケットに対して PEP が応答パケットを送信する実装例（以下，応答型 BiZA）の性能評価も示されている．応答型 BiZA では，SPA パケットを検証した PEP は認可結果を応答パケットとして UDP を用いて Agent に返信する．応答パケットは AES で暗号化されておりそのデータサイズは HMAC を含め 74 Byte である．一方 Agent はリスト 1 の Line1–6 の後，応答パケットの受信を待つ．そして，応答パケットを受信し，認可を確認できた場合にアクセス要求パケットを送出する．結果より，応答型 BiZA は BiZA より 0.06〜0.19 秒アクセス処理時間を短縮しており，TCP 接続の効率の観点では優れていることが分かる．

4.3.5 認可要求パケットのネットワーク負荷

認可要求パケットがネットワークに及ぼす負荷を測定するため，Web Proxy を Resource に設置し Subject からインターネットへの Web アクセス（著名な動画配信サイト，ポータルサイト，SNS など）を 30 分間繰り返し集中的に行った．**表 6** に結果を示す．この間 759 個のアクセス要求パケットおよび 137 回の認可処理が発生し，すべて「通過」となり接続失敗はなかった．また，認可失敗にともなうアクセス要求パケットの再送もなかった．TCP セッション数に対する認可要求パケット数の割合は 0.18 であり，短期間に連続的に発生する接続要求に対して効率的な認可処理を行えていることが分かる．また認可要求パケットに起因するパケット数・トラヒック量の増加は 0.2〜0.3%にとどまり負荷は軽微である．

ここで，複数台の端末が本実験時と同じ頻度で Web Proxy に通信する場合の認可要求パケット処理時間を推定する．

表 7 に認可要求パケットの発生と PEP での処理時間が M/M/1 の待ち行列に従うと仮定した場合の処理待ち時間の推定平均値および 90，99.9 パーセンタイル値を示

表 7 認可要求パケットの処理待ち時間推定値
Table 7 Estimated time of packet authorization for multiple hosts.

署名検証頻度	端末数	平均	90 percentile	99 percentile
0	500	6 msec	14 msec	28 msec
	1000	8 msec	18 msec	37 msec
	2000	21 msec	48 msec	98 msec
0.001	500	8 msec	14 msec	40 msec
	1000	12 msec	20 msec	145 msec
	2000	40 msec	82 msec	396 msec

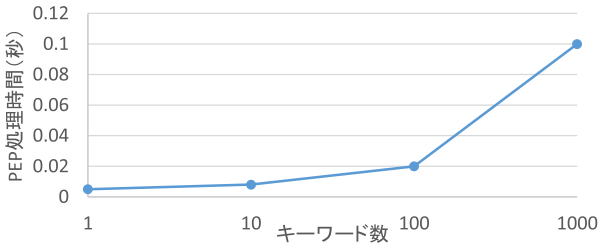


図 9 登録キーワード数と処理時間の関係
Fig. 9 Relation between keywords and PEP processing time.

す．署名検証頻度は AES 鍵を PEP と事前に共有していない Subject からのアクセス要求パケットの割合に比例する．1 度鍵が共有されれば，しばらくの間は署名検証が不要のため，頻度は 0.1%までを想定する．ここで PEP の限界性能を，処理待ち時間の 99 パーセンタイルと $P_{process}$ が一致する端末数と定める．$P_{process} = 200$ msec のとき，署名検証頻度が 0%，0.1%である場合の限界性能はそれぞれ 2,330 台，1,240 台となる．また同様の条件（処理待ち時間の 99 パーセンタイルが 200 msec）で，応答型 BiZA の限界性能はそれぞれ 2,050 台（検証頻度 0%），1,140 台（検証頻度 0.1%）となり BiZA と比べて 8〜12%低下する．理由としては PEP での応答パケットの作成に 0.6 msec を要することがあげられる．このため処理可能な端末台数の観点では BiZA の方が優れていることが分かる．

4.3.6 登録キーワード数と処理時間の関係

図 9 にアクセスポリシリストの監視項目#1，15，16 に記述したキーワード数と，PEP が 1 つの認可要求パケットを処理するのに要する時間との関係を示す．キーワード数の増加に比例して処理時間は長大化する．これは，BiZA の現実装では，各キーワードを順々に監視項目に対応する Bloom Filter に適用して検証処理を行うからである．

このため，現実装では高スループットが要求される PEP で，多数のキーワードをアクセスポリシスクリプトに登録するのは難しい．たとえば，表 7 と同条件下で接続端末数が 500 台，1,000 台のとき，設定可能な最大キーワード数は 73 件，25 件となる．よって，発生が確認されたばかりの脅威や高リスク（CVSS が 7.0 以上など）の脆弱性を含むソフトウエアやサービス，Subject 内で必ず動作するべきサービス，攻撃に悪用される恐れがあるツールなど，PEP の管理者が特に注目するキーワードを，要求スループットに応じ選択して登録するべきである．BiZA の利点の 1 つは Subject 内のセキュリティソフトウエア（アンチウイルス・EDR など）が対応できていない最新の脅威・脆弱性に対して，PEP 側で検出できることにある．登録後ほとんどの Subject で対応が完了したキーワードはアクセスポリシスクリプトから適宜外すことで，処理時間数を一定内に保つことが可能と考える．

また複数の CPU コアを利用できる PEP では検証処理を並列化することで時間短縮を図れる．Bloom Filter のアルゴリズム・実装の改良は今後の課題の 1 つである．

4.4 攻撃遮断性能

4.4.1 高速拡散マルウエアの遮断

Subject，PEP および PEP と同一端末で動作する Resource を同一の/24 サブネットに設置し，Subject にマルウエアを感染させ，PEP 上で遮断処理が可能か評価した．マルウエアとしては WannaCry，NotPetya を用いる．本実験に限り，安全性確保のため Subject も仮想マシン上で実行したが，実マシン上でも同等の結果が得られると考える．

マルウエアが有する攻撃先発見手段の 1 つにローカル TCP スキャンがある．ローカル TCP スキャンでは同一サブネット上を x.x.x.0，x.x.x.1，x.x.x.2，x.x.x.3…と小さいアドレス順にアクセスし，レスポンスがあった端末を SMB の脆弱性を悪用して攻撃する．実験ではマルウエアが Subject に感染してから PEP にスキャンを行うまでの時間が遮断に及ぼす影響を評価するために PEP の IP アドレスには x.x.x.20，x.x.x.50，xxx.230 の 3 種類を用いた．

アクセスポリシスクリプトは 2 通りを用いた．スクリプト A は監視項目#2 および#3 を用い「過去 300 秒以内に不審プロセス数 ≧ 5 またはスキャン先端末数 ≧ 20」の場合，遮断とする．スクリプト B は監視項目#1 を用い「既知の WannaCry，NotPetya バイナリのハッシュ値を持つプロセスが起動している」場合に遮断とする．

本実験で使用した WannaCry の検体 i[*1]は，実験環境下において起動後 42.8 秒でスキャンを開始する．/24 サブネットのスキャン完了には開始から 25.8 秒かかる．一方 NotPetya はスキャン完了には 900 秒を要する．また，起動後に自身のバイナリファイルをディスクから削除するという特性がある．

表 8 に実験結果を示す．スクリプト A に関しては，PEP アドレスが x.x.x.20 の場合 WannaCry，NotPetya ともに遮断に失敗するケースが発生した．原因はマルウエア挙動の更新速度がスキャン速度に追いつかなかったためである．特に WannaCry の拡散は高速であるため，複数回の試行すべてで遮断に失敗した．仮に $P_{update} = 1$ sec とすれば遮断

[*1] SHA-1 hash:e889544aff85ffaf8b0d0da705105dee7c97fe26

表 8 マルウエア遮断実験の結果
Table 8 Result of malware containment.

	PEP アドレス	WannaCry	NotPetya
スクリプト A	x.x.x.20	×	△
	x.x.x.50	○	○
	x.x.x.230	○	○
スクリプト B		○	×

○…遮断成功 ×…遮断失敗 △…タイミングにより成功・失敗

確度は向上すると考えられる．アドレスが x.x.x.50 以上では両マルウエアに対して遮断が成功する．このため，高速拡散型マルウエアに対して少なくとも 80% (= 1 − 50/255) 以上のアドレス帯にある Resource を保護できるといえる．スクリプト B に関しては，起動後に自身のバイナリの削除を行う NotPetya の遮断はできなかったが WannaCry に対する有効性は確認できた．すべてのマルウエアに対応できるスクリプトの設計は容易ではないが，複数の対策手段を重ね合わせることで遮断の精度を向上させることができると考える．

なお，NotPetya はスキャン以外にも Subject が確立中の TCP 接続の宛先端末や ARP キャッシュ情報を使った拡散も行う．先行文献では端末内にデセプション機能を設けることで，巧妙な拡散を発見する方法を検討している [35]．今後はこのような機能を監視項目に追加することも検討する．

4.4.2 APT 対する効果

MITRE が公開している標的型攻撃者 APT-29 のシナリオ Day-2 [36] では，悪性 PowerShell スクリプトを使用して攻撃を行う．シナリオでは初期感染端末から他端末に攻撃を拡散する．そこで，他端末への拡散前に初期感染端末で実行される悪性 PowerShell スクリプトを監視項目#8 で検出できるかを検証した．その結果，8 個の悪性スクリプトのうち，6 個を不審スクリプトとして検出できることを確認した．後述のとおり，一般業務で使用される端末上でも資産管理ソフトや端末管理ソフトが定期的に PowerShell スクリプトを実行される．しかしスクリプトの種類は限られるため，攻撃発生時とは異なり，資産管理ソフト・端末管理ソフトが原因で，これまで履歴のないスクリプトが短期間に複数種類にわたり実行されることは稀である．たとえば，実験に使用された端末を 4 カ月間分析したところ，上記ソフトは 1 カ月あたり平均 5.5 回の頻度で，新規のスクリプトをダウンロード・実行していた．しかし，同日（24 時間以内）に 2 件以上の新規スクリプトが起動されることはなかった．このため「過去 300 秒以内の不審スクリプト実行回数 $\geqq 2$」の場合遮断とするアクセスポリシで攻撃を遮断できる．

4.4.3 誤遮断について

BiZA で誤遮断が発生する主要因は以下の 2 点である．

1. Subject が悪意ある活動をしていないにもかかわらず，不審活動数が，アクセスポリシスクリプトに記した閾値を超える．
2. Subject 上で悪性プロセスが動作していないにもかかわらず，リスト型監視項目の Bloom Filter の False Positive によって，動作していると誤判定される．

表 9 監視項目と閾値
Table 9 Monitored activities and the thresholds.

	#2（閾値=5）	#3（閾値=20）	#8（閾値=2）
閾値未満	107,390	107,670	107,670
閾値以上	270	0	0
時間	47:34	00:00	00:00

1. に関して，**表 9** に，Subject から PEP に 26 日間の間に送られた認可要求パケット 107,670 個のうち 4.4.1 および 4.4.2 項で示した閾値を超えたものの数を示す．監視項目#3，#8 では閾値以上になるケースはなかった．一方監視項目#2 では閾値以上になる場合が 270 件あり合計 47 分間の誤遮断が発生する．これは全時間帯の 0.13%に相当する．なお#8 では閾値 = 1 とした場合，閾値を超えるアクセスは 107 件発生する．よって複数件の事象を基にリスクを測ることで誤遮断を減らすことができる．

2. に関して Bloom Filter を $P_{FP} = 10^{-6}$ の下で作成し PEP 上で 100 個のキーワードを検証対象とした場合，誤検知は 10^{-4} となる．このため 1 万台の端末が存在する組織で個々の端末で異なる Bloom Filter が作成される場合，誤遮断される端末数の期待値は 1 台となる．

1. と 2. のケースはともに誤遮断の頻度は許容範囲内と考える．また「一時遮断」の仕組みにより，誤遮断の悪影響を緩和できる．先述のとおり，アクセスポリシスクリプトは「リスク」の観点で通信の「通過」・「遮断」を行うものであり，複数端末間の相関分析をする [21] など，分析時間を要する検知処理の完了までの間，高リスクな Subject-Resource 間の通信を制限し被害拡大を防止するのが主目的でする．

しかし一方で，アクセスポリシスクリプト作成時には遮断効率・誤遮断の頻度のバランスを考え閾値を設定することが望ましい．既存のゼロトラストアーキの中には，一定期間 Subject の挙動を監視し閾値を設定する運用手順を示しているものがある [37]．また，過去のアクセス履歴を基にアクセスポリシを自動生成する手法も提案されている [38]．今後は上記技術を活用した制御方法を検討していく．

4.5 ユースケース・他技術との比較

4.5.1 脆弱性を有するプロセス起動への対応

SSH クライアントである PuTTY [39] にはいくつかの脆弱性が見つかっている．その中の 1 つである CVE-2019-17069 [4] は Ver.0.73 未満の PuTTY が有する脆弱性である．組織の重要 Resource を使用する Subject に対して，この脆弱性を持たない Ver.0.73，0.74（原稿執筆時点で最新）の PuTTY のみを許可するには，**図 10** に示すアクセスポリシスクリプトを記述する．このスクリプトでは Ver.0.73，

```
"effect": "allow",
"rules": { (略)
"context": { "$.mod_1": {"condition":"AnyIn",
"values":[
"fafe9c2c410aed31084c98892ad05237f709de81",
"d932604ab8e9debe475415851fd26929a0c0dcd1"]}
```

図 10 PuTTY を制御するアクセスポリシスクリプト
Fig. 10 Access policy script for PuTTY.

```
"effect": "allow",
"rules": { (略)
"context":{"$.mod_15": {"condition":"AllNotIn",
"values":["TermService"]}
```

図 11 RDP を制御するアクセスポリシスクリプト
Fig. 11 Access policy script for RDP.

```
"effect": "allow",
"rules": { (略)
"context":{"$.mod_16":
{"condition":"AnyOf","values":[
{"condition":"AnyIn","values":["KB4512497"]},
{"condition":"AnyIn","values":["KB4512517"]},
{"condition":"AnyIn","values":["KB4512507"]},
 (略)
```

図 12 HotFix アクセスポリシスクリプト
Fig. 12 Access policy script for HotFix.

0.74 とハッシュ値が一致するプロセスが起動している場合のみ接続を許可する．なお，別の方法として Ver.0.72 未満のハッシュ値を記述し接続を禁じることもできる．どちらのアプローチをとるかは，記述する必要があるハッシュ値の数などに依存する．

4.5.2 脆弱性を持つサービスへの対応

BlueKeep [41] のような深刻な脆弱性が発生した場合，パッチを適用するまでの緩和策として，一時的に関係するサービス（この場合 RDP）を停止するという手段がある [42]．**図 11** にスクリプト例を示す．このスクリプトでは Subject が Term Service サービスを起動していない場合に限りアクセスを許可する．また HotFix が提供されている場合は**図 12** に示すように KB 番号を指定してアクセス制御を行うこともできる．ただし，Windows 上では大規模アップデートのときなどに複数の KB が 1 つの KB にまとめられて番号が変わることがあるため，スクリプト記述の際は注意が必要である．

4.5.3 内部犯行への対応

IPA の調査によると，内部犯行に起因の情報漏洩における情報持ち出し手段は，1 位：USB メモリ，2 位：スマートフォン，3 位：電子メール，4 位：紙媒体，5 位：Web アップロードとなっている（顧客情報の場合）[24]．BiZA では挙動監視モジュール#9，#10，#11 で，1 位，4 位，5 位の手段に対応できる．これらの手段を使った犯行の割合は 50.7%であり，過半数の持ち出し手段に対応できることが分かる．

実行ファイルやシナリオが公開されているマルウエアや標的型攻撃とは異なり，内部犯行について定量的な評価は容易ではない．シナリオを定めたより詳細な評価は今後の研究課題とする．

4.5.4 技術比較

表 10 に BiZA と他技術との比較を示す．一般的なゼロトラストアーキテクチャと比較すると，SPA を拡張した BiZA は認可粒度およびリアルタイム性や通信負荷の点で優れている．Poise，EzTrust との比較では，認可粒度や処理時間などの点では劣っているものの，リスト型挙動情報，不審挙動情報を活用した柔軟なアクセス制御が可能な点，認可要求パケットの暗号化・検証をサポートしている点，PEP 設置の汎用性などの点で優位である．一方で他の外部サービスとの連携などについては今後の研究開発課題である．また認可対象ユーザに使用できる識別子が端末ログインアカウントに限られる点も今後改善が必要である．

5. 考察および今後の課題

BiZA は既存のゼロトラスト技術と比較して，リアルタイム性が高い多様な挙動情報を用いた柔軟な動的認可を TCP 通信全般に適用できるのが利点である．今後の課題の 1 つとして TCP に加え UDP フローにも対応していくことがあげられる．またより多くの端末・長期間の評価を行いスケーラビリティ・信頼性の検証を行っていく．

また SPA およびその派生形に共通の課題として，認可要求パケットが認可されたタイミングで被認可端末と同じ IP アドレスを持つ別端末からアクセス要求パケットを送信すると，通信が通過してしまうという問題がある [15], [43]．これは NAT 下にある Subject がインターネット上の PEP・Resource と通信する際に特に大きなリスクとなりうる．解決には認可要求パケットとアクセス要求パケットとの確実な紐づけが必要である．対策としては，共通鍵とアクセス要求パケットの Initial Sequence Number（ISN）などを基に HMAC を作成し，IP ヘッダの識別子フィールドなどに埋め込む，などの方法が考えられる．今後実装・検証を行う予定である．

また Agent 自体が攻撃対象になる可能性がある．これに対する解決策としては Agent プロセスの実行権限に制約をかけ本来想定していない動作を極力禁じるなどのハードニングが考えられる [44]．

次に 3.6 節の式 (3) で示した interval 算出方式の妥当性について考察する．4.3.4 項および 4.3.5 項の実験において，PEP の処理完了前にアクセス認可パケットが送付され認可に失敗する事象はなかった．また表 5 に示したとおり，interval をつねに同一値に設定した場合と比べると，提案方式はアクセス処理時間の短縮に成功している．以上より，実験に用いた小規模環境であれば，式 (3) は一定の妥当性を持つと考える．一方で，応答型 BiZA と比べると

表 10 他技術との比較
Table 10 Comparison with other techniques.

	BiZA	SPA[11]	一般的なゼロトラストアーキテクチャ([2][3][4][5])	Poise[6]	EzTrust[7]
挙動情報のリアルタイム性	数秒単位	—	↓（数分単位[5]～）	—	—
一度の認可に係るパケット数	１パケット	—	↓（複数パケット．認可毎に PEP・PDP 間，又は Subject・PDP 間で双方向通信が発生）	—（1 パケット）	↑（0 パケット．埋込み型のため）
認可にかかる時間	5msec～	—	↓（PDP 経由の認可のため通信遅延大）	↑（数 μ sec）	—
認可粒度	TCP セッション単位	—	↓（ユーザが Resource 上のアプリケーション又は PDP にログインするタイミング．その後一定時間は再認可無し）	↑（任意のタイミングで再認可が可能）	↑（パケット単位）
過去履歴に基づく不審挙動の監視	プロセス・通信・PowerShell	↓（なし）	—(連携する EDR, AV に依存)	↓（なし）	↓（なし）
リスト型挙動情報への対応	プロセスリスト，サービスリスト，HotFix リスト	↓（なし）	—（実装可能）	↓（なし）	↓（なし）
外部サービスとの連携	今後検討	—	↑（セキュリティベンダや IDaaS 提供者などとの連携）	—	—
認可対象ユーザ識別子	端末ユーザ（もしくは設定ファイル等で指定）	—	↑（複数のアカウントから自由に指定が可能）	—	—
認可パケットの暗号化	ECDSA, HMAC	—	—	↓（なし）	↓（なし）
認可パケットの認証	RSA, AES	—	—	↓（なし）	↓（なし）
PDP への負荷	低（動的情報に基づく認可が PEP で完結のため）	—	↓（認可毎に PDP へ問合せが必要）	—	—
PEP の汎用性	スイッチ・エンドポイントのどちらにも, Resource 上のアプリケーションの改変無く設置可能	—	↓（ゲートウェイ上に設置，もしくは Resource 上のアプリケーション上で SSO 対応等が必要）	↓（P4-Programmable Switch 上のみ）	↓（同一計算機上のマイクロサービスの間のみ）

(↑：BiZA より優位．↓：BiZA より劣位．－：BiZA と同等)

処理時間は増えており，算出方式の改良の余地はあると考える．また，1,000 台の端末が同時接続するような大規模環境での実機検証についても今後の課題としたい．

BiZA の現実装では PEP はすべての Subject の公開鍵をあらかじめ保持していることを前提としている．文献 [45] では IoT デバイス向けの軽量な X.509 証明書を検討しておりデータサイズを 146 Byte まで圧縮できると述べている．今後はこのような軽量証明書の活用も検討していく．

6. おわりに

本稿では SPA を拡張してリアルタイム性が高い動的情報を活用した認可を，保護対象ネットワーク内で発生するすべての TCP 通信に効率的に実現するゼロトラストアーキテクチャ BiZA の設計について述べた．BiZA は Bloom Filter を用いて 16 種類の挙動情報を 1 つの UDP パケットに格納し，送信間隔を調整することで通信効率を上げる．評価実験を通じて WannaCry，NotPetya など高速拡散型マルウエアにも追随して被害拡大を抑止できることを検証したほか，IoC と連携した事前対策，脆弱性対策，内部犯行にも適用できることを検証した．認可処理がネットワークトラヒックに及ぼす影響は 0.2～0.3%と軽微であることを確認した．

今後は，多数の端末に対する長期間の評価，アクセスポリシスクリプトの自動生成，UDP フローへの対応，SPA に起因する課題の解決などを通じて，アーキテクチャの改良を進めていく．

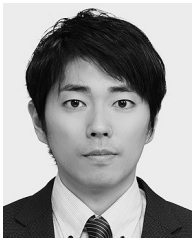
参考文献

[1] Rose, S., Borchert, O., Mitchell, S. and Connelly, S.: NIST SP800-207: Zero Trust Architecture, available from ⟨https://csrc.ist.gov/publications/detail/sp/800-207/final⟩ (accessed 2021-05-02).
[2] Spear, B., Beyer, B.A.E., Cittadini, L. and Saltonstall, M.: Beyond Corp: The Access Proxy, *login:*, Vol.41, No.4, pp.28–33 (2016).
[3] Microsoft Azure：条件付きアクセスとは, 入手先 ⟨https://docs.microsoft.com/ja-jp/azure/active-directory/conditional-access/overview⟩（参照 2021-05-02）.
[4] Cisco Duo セキュリティ, 入手先 ⟨https://www.cisco.com/

c/dam/global/ja_jp/products/catalog/pdf/cisco-duo-security-brochure.pdf⟩（参照 2021-05-02）.
[5] Appgate. Appgate SDP -PERFORMANCE AND SCALING (2020).
[6] Kang, Q., Xue, L., Morrison, A., Tang, Y., Chen, A. and Luo, X.: Programmable In-Network Security for Context-aware BYOD Policies, *Proc. 29th USENIX Security Symosium*, pp.595–612 (2020).
[7] Zaheer, Z., Chang, H., Mukherjee, S. and Van der Merwe, J.: eZTrust: Network-Independent Zero-Trust Perimeterization for Microservices, *Proc. ACM SOSR'19*, pp.49–61 (2019).
[8] OConor, T.J., Enck, W., Petullo, W.M. and Verma, A.: PivotWall: SDN-Based Information Flow Control, *Proc. ACM SOSR'18*, pp.1–14 (2018).
[9] DeCusatis, C., Lienghtiraphan, P., Sager, A. and Pineli, M.: Implementing Zero Trust Cloud Networks with Transport Access and First Packet Authentication, *IEEE Int'l Conference on Smart Cloud 2016* (2016).
[10] Rash, M.: Single Packet Authorization, *Linux Journal* (Apr. 2007).
[11] Rash, M.: Protecting SSH Servers with Single Packet Authorization, *Linux Journal* (May 2007).
[12] Cloud Security Alliance：Software-Defined Perimeter アーキテクチャガイド，入手先 ⟨https://www.cloudsecurityalliance.jp/site/wp-content/uploads/2020/03/sdp_architecture_guide_v2_J_FINAL.pdf⟩（参照 2021-05-02）.
[13] Moubayed, A., Refaey, A. and Shami, A.: Software-Defined Perimeter (SDP): State of the Art Secure Solution for Modern Networks, *IEEE Network*, September/October 2019, pp.226–233 (2019).
[14] fwknop, available from ⟨https://github.com/mrash/fwknop⟩ (accessed 2021-05-02).
[15] Krmelj, G.R., Pancur, M., Grohar, M. and Ciglaric, M.: OpenSPA – An Open Extensible Protocols for Single Packet Authorization, *Proc. ACM CECC 2018* (2018).
[16] SPA (Single Packet Authorization) 解説, 入手先 ⟨https://cloudsecurityalliance.jp/newblog/2019/08/27/448/⟩（参照 2021-05-02）.
[17] Panda Security: #WannaCry Report, available from ⟨https://www.pandasecurity.com/mediacenter/src/uploads/2017/05/1705-Informe WannaCry-v160-en.pdf⟩ (accessed 2021-05-02).
[18] Cylance：Petya-Like ランサムウエアは厄介なワイパー, available from ⟨https://www.cylance.com/ja jp/blog/jp-threatspotlight-petya-like-ransomware-is-nasty-wiper.html⟩（参照 2021-05-02）.
[19] 畠山昴大, 小谷大祐, 岡部寿男：ゼロトラスト認証認可連携におけるユーザ同意付きコンテキスト共有, DICOMO2020 シンポジウム予稿集，pp.640–647 (2020).
[20] Hypertext Transfer Protocol Version 3 (HTTP/3), draft-ietf-quic-http-34, available from ⟨https://tools.ietf.org/html/draft-ietf-quic-http-34⟩ (accessed 2021-05-06).
[21] 川口信隆，築地原護，井手口恒太，谷川嘉伸，冨村英勤：不審活動の端末間伝播に着目した標的型攻撃検知方式，情報処理学会論文誌，Vo.57, No.3, pp.1022–1039 (2016).
[22] 藤井翔太，川口信隆，重本倫宏，山内利宏：Cyber Threat Intelligence の構造化による分析支援手法の提案，情報処理学会研究報告，Vol.2021-CSEC-92, pp.1–8 (2021).
[23] Symantec: Living off the Land Turning Your Infrastructure Against You, available from ⟨https://docs.broadcom.com/doc/living-off-the-land-turning-your-infrastructure-against-you-en⟩ (accessed 2021-05-05).
[24] 情報処理推進機構：内部不正による情報セキュリティインシデント実態調査，入手先 ⟨https://www.ipa.go.jp/files/000051140.pdf⟩（参照 2021-05-05）.
[25] Bloom, H.: Space/time trade-offs in hash coding with allowable errors, *Comm. ACM*, Vol.13, No.7, pp.422–426 (1970).
[26] RobWillis.info: Everything You Need To Know To Get Started Logging PowerShell, available from ⟨https://robwillis.info/2019/10/everything-you-need-to-know-to-get-started-logging-powershell/⟩ (accessed 2021-05-07).
[27] MITRE ATT&CK, available from ⟨https://attack.mitre.org/⟩ (accessed 2021-05-08).
[28] Kumar, S.S.: An emerging threat Fileless malware: A survey and research challenges, *Cybersecurity*, Vol.3, pp.1–12 (2020).
[29] Bohannon, D.: Revoke-Obfuscation: Powershell Obfuscation Detection Using Science, BlackHat USA (2017).
[30] NIST SP800-57 Part1 Revision 5 Recommendation for Key Management, available from ⟨https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf⟩ (accessed 2021-05-10).
[31] Py-ABAC, available from ⟨https://github.com/ketgo/py-abac⟩ (accessed 2021-05-14).
[32] PyDivert, available from ⟨https://pypi.org/project/pydivert⟩ (accessed 2021-05-16).
[33] Pycryptodomex, available from ⟨https://pypi.org/project/pycryptodomex/⟩ (accessed 2021-05-16).
[34] Kuchler, A., Mantovani, A. and Han, Y.: Does Every Second Count?, Time-based Evolution of Malware Behavior in Sandboxes, *Proc. NDSS2021* (2021).
[35] 川口信隆：TCP Connection Table を悪用した組織内ネットワークへのマルウエア拡散の特性評価と対策検討，情報処理学会論文誌，Vol.61, No.9, pp.1428–1443 (2020).
[36] Mitre-attack/attack arsenal, available from ⟨https://github.com/mitre-attack/attack-arsenal/tree/master/adversary_emulation/APT29/Emulation_Plan/Day%202⟩ (accessed 2021-05-23).
[37] Peck, J., Beyer, B., Beske, C. and Saltonstall, M.: Mitigating to BeyondCop: Maintaining Productivity While Improving Security, *login:*, Vol.42, No.2 (2017).
[38] Sanders, M.W. and Yue, C.: Mining Least Privilege Attribute Based Control Policies, *Proc. ACSAC'19*, pp.404–416 (2019).
[39] PuTTY, available from ⟨https://www.PuTTY.org/⟩ (accessed 2021-05-23).
[40] CVE-2019-17069, available from ⟨https://nvd.nist.gov/vuln/detail/CVE-2019-17069⟩ (accessed 2021-05-23).
[41] CVE-2019-0708, available from ⟨https://nvd.nist.gov/vuln/detail/cve-2019-0708⟩ (accessed 2021-05-23).
[42] Welivesecurity: It's time to disconnect RDP from the internet, available from ⟨https://www.welivesecurity.com/2019/12/17/bluekeep-time-disconnect-rdp-internet/⟩ (accessed 2021-05-23).
[43] Zorkta, H. and Almutlq, B.: Harden Single Packet Authentication (HSPA), *Int'l. Journal of Computer Theory and Engineering*, Vol.4, Nol.5 (2012).
[44] Mirheydari, M., Zavarsky, P. and Butakov, S.: Single Packet Authorization in a Multi-layered Security Architecture, *IEEE 29th Biennial Symposium on Communications* (2018).
[45] Papadimitraos, P. and Raza, S.: Lightweight X.509 Digital Certificates for the Internet of Things, *InterIOT 2017* (2017).
[46] Download.NET Core 3.1, available from ⟨https://dotnet.microsoft.com/download/dotnet/3.1⟩ (accessed 2021-09-08).

川口 信隆 （正会員）

2008 年 3 月慶應義塾大学大学院理工学研究科開放環境科学専攻後期博士課程修了，博士（工学）．2008 年 4 月より株式会社日立製作所にてサイバーセキュリティ，情報セキュリティの研究開発に従事．コンピュータセキュリティ研究会専門委員，電子情報通信学会論文誌編集委員．IEEE 会員．CISSP．



西嶋 克哉

2014 年早稲田大学大学院先進理工学研究科電気・情報生命専攻修士課程修了．同年 IT ベンダに入社しエンジニアとして従事．2018 年 7 月より株式会社日立製作所にてサイバーセキュリティ，情報セキュリティの研究開発に従事．CISSP．

重本 倫宏 （正会員）

2006 年大阪大学大学院基礎工学研究科システム創生専攻修士課程修了．同年（株）日立製作所システム開発研究所（現，システムイノベーションセンタ）入所．現在はネットワークセキュリティ技術に関する研究開発に従事．博士（工学）．