ゼロトラスト認証認可連携における異なるデータソース間のコンテキストの紐づけ

平井 雅人[1] 小谷 大祐[1] 岡部 寿男[1]

概要：コンテキストと呼ばれるユーザーやデバイスの性質や状況を示す情報を用いて認可を行うゼロトラストアーキテクチャ (ZTA) というアクセス制御モデルが注目されているが、ZTA を利用して認証認可連携を行う手法として、ゼロトラスト認証認可連携 (ZTF) が提案されている。ZTF はコンテキストを様々な収集元 (データソース) を経由して集めることでこれを行う。しかし、ZTF では拡張実装が容易でないデータソースからのコンテキストの収集手法は明確でなかった。そこで、本研究では ZTF におけるコンテキストの汎用的な収集手法として、コンテキストをエンドユーザーから直接収集するエンティティ (CtxC) とコンテキストを組織に提供するエンティティ (CAP) に分けて考え、CAP が CtxC のコンテキストの識別子を CAP の識別子で対応づける方法について提案する。CtxC での拡張実装が容易な場合では CAP との間で仮名 ID を共有して行い、そうでなく CtxC と CAP の管理者が同一の場合は管理者が対応表を作っておく。また、証明書を利用して認証することで管理者が不正な紐付けを行わないことを信頼することなく対応づける手法についても提案する。そして、RADIUS や Intune を CtxC として紐付けの実装についても行い、対応を紐付けられることを確認した。

Linking Contexts from Distinct Data Sources in Zero Trust Federation

Masato HIRAI[1] Daisuke KOTANI[1] Yasuo OKABE[1]

1. はじめに

組織がネットワーク上の情報資産を管理する手法としてゼロトラストアーキテクチャ (ZTA)[10] と呼ばれるアクセス制御モデルが注目されている。ZTA ではアクセス元の情報を利用してアクセス元を常に検証し検証結果を用いてアクセスの認可を行う。ZTA で検証に利用される情報はコンテキストと呼ばれている。コンテキストにはユーザーやデバイスの位置情報やアクセス履歴などのアクセス元の周辺状況や行動履歴などの常に移り変わりうる情報が含まれる。

ZTA では、組織において認可に必要なコンテキストを全て自組織で収集する設計が一般的である。しかし、Identity Federation(IdF) のように複数組織で認証認可を連携する場合、この設計では他組織で収集されたコンテキストを利用することができず、十分な認可判断を行うのは困難である。そこで、ZTA を拡張して IdF で複数の組織のコンテキストを利用して認可判断を行う手法としてゼロトラスト認証認可連携 (Zero Trust Federation; ZTF) が提案されている [4]。ZTF では組織を横断してコンテキストを共有する仕組みが提案され、この共有を行うエンティティとして Context Attribute Provider(CAP) が定義されている。CAP は組織と独立にコンテキストを収集し、ユーザーの認可のもと各組織 (Relying Party; RP) にコンテキストを提供する。具体的には、UMA(User Managed Access)[6] を利用したユーザの認可によるアクセス制御に基づき、RP が CAP と CAEP(Continuous Access Evaluation Protocol)[12] でコンテキストを相互に共有するという手法である。しかし、CAP が RP 以外から独自にコンテキストを収集する場合について、記述はされていたが具体的な方法は明確ではなかった。ZTF においてより確度の高い認可判断を行うためには、より多くのデータソースから多様なコンテキ

[1] 京都大学
Kyoto University

ストを収集する必要がある。そのため、CAP が独自に RP 以外のデータソースからコンテキストを収集する場合についても考慮する必要がある。

コンテキストをユーザーやデバイスから収集できるデータソースは多様で、Web アプリケーションにおけるアクセスの挙動についてのログや、組込みシステムのようなもので集められるログなど、その性質は様々である。RP と同様の手法でデータソースが CAP とコンテキストの共有を行うためには、認可サーバーと通信してユーザーの認可状況を取得し、適切なユーザーやデバイスの識別子を付与して CAP にコンテキストを送信するなどの多くの実装が必要である。これらの実装は全てのデータソースで可能とは限らず、特に長く運用されている組込みシステムのように後から追加で大きな実装を行うことが困難な場合においてはこの手法を利用することは難しい。そのような個別のケースで個々に設計を行っていると実装にかかるコストが大きくなり、多様なデータソースからコンテキストを収集することが現実的に難しくなる。

そこで本研究では、コンテキストの収集方法について分類し、考察することで CAP として望ましいアーキテクチャを示し、特に考慮が必要な CAP との連携のための追加実装が困難なデータソースにおける設計方法を提案する。

本研究ではコンテキストをユーザーやデバイスから直接収集するエンティティを Context Collector(CtxC) として定義し、CtxC が集めたコンテキストを CAP に渡す方法について論じる。CtxC によって収集されたコンテキストは通常 CtxC 独自のユーザーやデバイスの識別子 (CtxC-id) が含まれる。CAP が CtxC から受け取ったコンテキストをユーザーの認可のもと各組織へと提供するには、そのコンテキストが CAP における CAP-id で識別されるどのユーザーやデバイスに対応するのかを判別できなければならない。従って問題となるのは、どのようにコンテキストに含まれる CtxC での CtxC-id と CAP での CAP-id を対応づけるかである。本研究ではこれをコンテキストの紐付けと呼び、この問題について取り組む。

本研究では、(1) 従来の ZTF で提案されていた CtxC の追加実装が容易なケース、(2) 管理者が行うことで追加実装を必要としない CtxC と CAP の管理者が同一であるケース、(3)CtxC がユーザーやデバイスの証明書を認証に利用している、管理者が不正な紐付けを行わないことを信頼する必要のないケースの 3 つのケースにおいて、CAP が識別子の対応づけを行うための設計手法を提案する。(1) では事前に CtxC と CAP の間で仮名 ID を相互に交換し、CtxC が仮名 ID をコンテキストに含めてから CAP に送ることで CAP がコンテキストを対応づける。(2) では管理者が直接確認するなどの手段で CtxC と CAP での識別子の対応を確認し、その対応表を CAP に設置することで対応づける。(3) では CAP がユーザーやデバイスに CtxC で利用している証明書を提示させ、その認証によって対応づける。

また、本研究では (3) のケースにおける実装として、デバイスのどの LAN に接続しているか、LAN での通信ログ、デバイスの OS に既知の脆弱性がなく十分新しいかなどのような要素を検証することで認証認可を行うことを想定し、802.1X の EAP-TLS で認証を行う RADIUS サーバーを CtxC とする実装と Microsoft の MDM サービスである Intune を CtxC とする実装の二つの実装を示す。これらの実装を通して、証明書を利用してコンテキストの識別子の対応づけを行うための具体的な実装方法を示す。

本研究の構成は次の通りである。第 2 章では本研究の背景となる技術について説明し、第 3 章では CAP の分類や課題の解決手法の説明を行う。第 4 章では RADIUS サーバーのアカウンティングログと Intune を用いた CAP のプロトタイプを用いてユースケースとともに説明する。第 5 章では第 3 章と第 4 章を通して得られた知見を整理し、考察する。最後に第 6 章で本研究をまとめる。

2. 関連研究

2.1 ゼロトラストアーキテクチャ (ZTA)

ZTA は、従来のアクセス制御手法である、組織の境界で脅威を排除することで境界内に暗黙の信頼を置く境界型と対比される新しいアクセス制御のモデルである。ZTA ではそのような暗黙の信頼を全て取り除くことを目的に、常にアクセス要求を検証することでアクセス制御を行うモデルである。

ZTA の検証の際には、コンテキストと呼ばれるユーザやデバイスが置かれた状況のような逐次移り変わる情報やユーザー、デバイスに関する静的な情報を用いる。ZTA で利用されるコンテキストの例として、アクセス履歴や位置情報などが挙げられる。普段と異なるパターンでアクセスされている場合、ユーザーやデバイスの行動が不審であると考えることができる。このようにコンテキストを利用することでアクセス元が信頼できるかどうかを常に検証し、アクセス制御を行うことができる。

このコンテキストを収集するために、組織内にあるリソースへの全てのアクセスの監視、記録が行われる。具体的な実装として、ユーザからの全てのアクセスを仲介するプロキシを設置し、このプロキシによって一元的にコンテキストの収集やアクセス制御を行うものがある [5]。

2.2 Identity Federation(IdF)

identity とはユーザに関する様々な情報のことであり、識別子やユーザの年齢などの属性が含まれる。この identity を複数の組織間で連携する仕組みが Identity Federation(IdF) である。IdF では Identity Provider(IdP) という IdF 内のユーザーの identity を管理し、IdF 内の組織に提供するエ

ンティティがある。IdF に参加している組織では IdP で登録している identity を利用できるため、ユーザは IdF に参加しているサービス 1 つ 1 つにユーザ登録を行う必要がない。これによって、ユーザーはシームレスに各サービスにアクセスできるようになる上、管理するアカウントが減少することでユーザビリティが向上すると期待される。

IdF を構築する技術として SAML[7] や OpenID Connect(OIDC)[8] などがある。OIDC では、identity を IdP から受け取ってサービスのために利用するエンティティを Relying Party(RP) と呼んでおり、本研究においても IdF におけるクライアントを RP と呼ぶ。

2.3 ゼロトラスト認証認可連携 (Zero Trust Federation; ZTF)

ZTF とは、Identity Federation(IdF) 内の RP が認証や認可の判断を行う際に、ZTA のようにコンテキストを用いるための枠組みのことである [4]。2.1 節で ZTA は単一の組織が一元的にコンテキストを収集していた。しかし、IdF のように複数の組織が連携する場合では、自組織で収集可能なコンテキストのみでは不十分である。

そのため、ZTF を実現するにはコンテキストを他組織からも収集する必要がある。この際に IdF ごとにコンテキストを集めるように設計してしまうと、ユーザーの利用頻度が低い IdF では認可に利用できるコンテキストが少なくなってしまう懸念がある。これらの問題に対処するために、ZTF は Context Attribute Provider(CAP) というエンティティを定義し、CAP が IdF を横断してコンテキストの収集と提供を行うことで、ユーザーの利用頻度が低い IdF でも十分な種類、量のコンテキストを用いて認証認可を行うことができる枠組みとして提案されている。

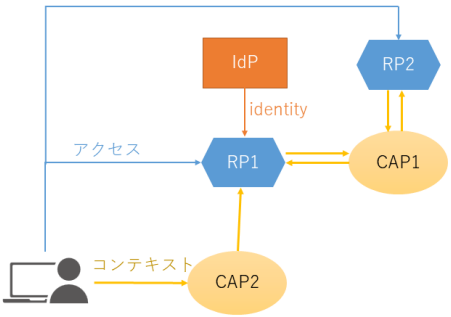

図 1　ZTF の概要図

図 1 で ZTF の概要を説明する。CAP1 は RP からコンテキストを受け取る CAP である。この CAP は他の RP からもコンテキストを受け取り、それを共有している。CAP2 はユーザーやデバイスから直接コンテキストを収集し、RP へと提供する CAP である。

CAP がコンテキストを RP に提供するためのプロトコルとして、継続的に認証認可を行うための Continuous Access Evaluation Protocol(CAEP)[12] とユーザーの認可のもとでアクセスを認可するための User Managed Access(UMA)[6] を組み合わせたプロトコルが提案されている [4]。CAP が RP からコンテキストを受け取ることについても同様の手法で実現される。

しかし、ZTF においては図 1 の CAP2 のような RP 以外からのコンテキストの収集方法について明らかではなかった。ZTF においては、多様なコンテキストを収集することが認可の質に直結するため、RP 以外からもコンテキストを収集することが求められる。例えば、入退室に用いる IC カードの利用履歴をコンテキストとして利用することで、ユーザーがどのような行動を行っていたかを知ることができる。一方でこの入退室記録を取得するシステムは、通常 RP のように認証のためにユーザーに HTTPS などのプロトコルで直接アクセスされる手段を提供せず、UMA の認可サーバー [6] と通信して認可された CAP からのアクセスかを確認して CAP に送信するといった機能も提供しない。追加で実装を行うことでこれらを行うことは可能かもしれないが、組込みシステムなど現実的でない場合も考えられる。そのため、RP と同様のプロトコルをそのまま用いることはできない。

3. コンテキストの紐付け

3.1 Context Collector(CtxC) の定義

ZTF において、CAP はコンテキストを何らかの手段で収集し、RP に提供するエンティティである。コンテキストはユーザーやデバイスを取り巻く状況により逐次変化する動的な情報が想定され、RP や CAP で収集されるとされている。

しかし 2.3 節のように、コンテキストを直接集められるデータソースの中には [4] に従った方法で認可などをしながら RP にコンテキストを提供することが難しいものも考えられる。例えば、RADIUS サーバーというデバイスの認証と通信記録を行うサーバーが、ユーザーの認可を確認して RP に直接コンテキストを送信するように追加で実装を行うことは困難である。もしこれを行うとすると、RADIUS サーバーでユーザーを認証するために HTTPS などで直接通信する機能や、RP や認可サーバーと独自に通信してコンテキストへのアクセス要求の正当性を判断し、コンテキストを RP へと送信せねばならないことになる。しかし、本来の RADIUS サーバーの役割と大きく異なる機能を実装し、管理することは管理者にとって大きな負担となる。また、コンテキストを直接収集するエンティティは数多く、そのそれぞれに同様の実装と管理を行うのは非常に大きなコストがかかると考えられる。従って、コンテキストの収集と提供の両方を行う単一のエンティティを前提とすることは適切ではない。

そこでより望ましいアーキテクチャを考えるために、こ

の例の RADIUS サーバーのようなコンテキストを直接収集するエンティティを Context Collector(CtxC) として定義することで CAP の役割を明確にする。これにより、CtxC がコンテキストを収集する役割を担い、CAP がユーザーの認可を確認し、RP にコンテキストを提供する役割を担うことになる。また、以下で CAP はコンテキストを必ず間接的に CtxC から収集するものと再定義する。

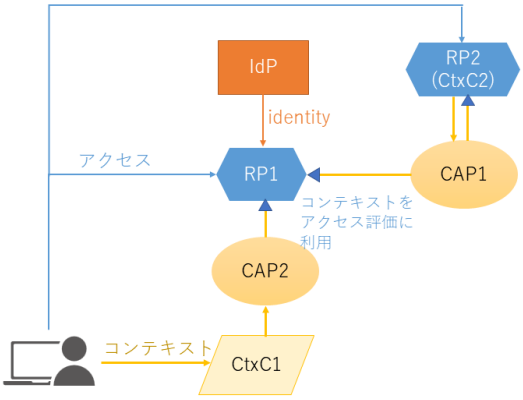

図 2 CtxC を導入して整理した ZTF

このことについて図 2 を用いて説明する。図 2 は図 1 に CtxC を追記したものである。この図から分かる通り、CAP2 はコンテキストを CtxC1 から受け取っている。また、CAP1 はコンテキストを RP から受け取っているが、コンテキストを受け取るという文脈において、RP は CtxC とみなす。

3.2 コンテキストの紐付け

CtxC でコンテキストを収集すると、通常そのコンテキストは CtxC 独自のユーザー/デバイス識別子 (CtxC-id) で識別される状態にある。その CtxC-id で識別できるコンテキストを受け取った CAP は、RP がコンテキストを利用することをユーザーが認可したのかを管理するための CAP でのユーザーの識別子 (CAP-id) と CtxC-id の対応を調べなければならない。どのユーザーがどの RP に対して認可を行ったのかについては、OAuth2.0[3] や UMA[6] などのプロトコルでアクセストークンを通じて得ることができる。従って、CAP は CtxC-id と CAP-id の対応を得なくてはならないことになる。このように、コンテキストに含まれる CtxC-id からコンテキストと CAP-id の対応を得ることをコンテキストの紐付けと呼び、本研究ではこの問題の解決に取り組む。この紐付けは確実で安全な手法で行う必要がある。この問題を CtxC や CtxC-id に何の仮定も付さずに解決することは難しいため、CtxC の拡張実装が容易な場合、CtxC と CAP の管理者が同一でない場合、CtxC が証明書で認証を行う場合の 3 つの場合についてそれぞれ解決手法を以下で提案する。

3.3 CtxC の拡張実装が容易な場合の紐付け方法

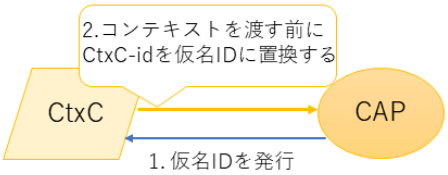

図 3 CtxC の拡張実装が容易な場合

Web アプリケーションのように CtxC の拡張実装が容易な場合では、事前に CAP から CtxC に発行された識別子を CtxC がコンテキストに含めて CAP に送る。

具体的にコンテキストを紐付ける方法を図 3 を用いて説明する。まず、事前に CAP が仮名の識別子 (仮名 ID) を CtxC に発行する。この時 CAP は仮名 ID と CAP-id の対応を保存し、CtxC は仮名 ID と CtxC-id との対応を保存する。

仮名 ID の発行は、例えば OpenID Connect[8] の ID トークンを用いることで実装できる。その後、CtxC はコンテキストに含まれる CtxC-id を仮名 ID に置換して CAP に送信する。この一連の手続きによって、CAP は自身にとって既知の識別子である仮名 ID が付与されたコンテキストを受け取ることができ、容易にコンテキストを紐付けることができる。

またこの手法とは逆に、CtxC が CAP に仮名 ID を発行することも可能である。手法についてはほとんど同様であるため省略する。どちらの場合でも CtxC での大きな拡張実装は必要である。

3.4 CtxC の拡張実装が容易でない場合の紐付け方法

この節以降では CtxC の拡張実装が容易でない場合について考慮してコンテキストを紐付ける手法について論じる。この場合では、組込みデバイスなどを想定し、現状の実装に関わる形の拡張実装を行うことができないとして考える。前節のように CtxC と CAP で仮名 ID を CAP と共有するには、CtxC がユーザーからブラウザなどで直接アクセスされることでユーザーを CtxC-id で認証する必要がある。これを行うには CtxC の認証についての拡張的な実装が必要であり、これは条件に適さない。

従って本研究では、CtxC はコンテキストを CAP に安全に伝達する手段のみを追加し、それ以外の CtxC での実装は行う必要がないように設計を行った。

3.4.1 CtxC と CAP の管理者が同一の場合の紐付け方法

図 4 に示すように、この場合においては、CtxC-id と CAP-id の対応を管理者が知っているので、その対応表を CAP に設置すればよい。CAP はこの対応表をもとに、コンテキストの紐付けを行う。

図 4　管理者が CtxC-id と CAP-id を紐付ける

例えば、企業において IC カードによる入退室管理システムを CtxC とする CAP が運用されているとする。この時、入社時の IC カードの登録の際に、IC カードの識別子と社員 ID の対応を管理者が作成し、CAP に登録することで、CAP はコンテキストを容易に紐付けることができる。

この手法であれば、CtxC から CAP に何らかの方法でコンテキストを伝送するだけでよく、ほとんど追加実装は必要ない。ただし、常に管理者による管理を必要とするため、管理者を信頼しなければならない。

3.4.2　CtxC が証明書を用いた認証を行う場合の紐付け方法

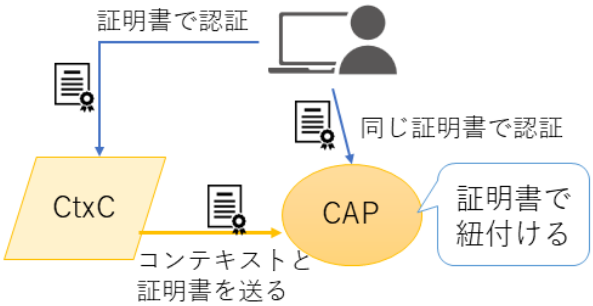

図 5　証明書で公開鍵認証する場合の概要

3.4.1 節の方法は攻撃者に協力した管理者が攻撃者にとって都合の良いコンテキストを紐付けるために不正な対応表を作成するなどの不正行為を行わないことを暗黙的に信頼する必要があるという問題がある。そこで、CtxC が証明書で認証し、CtxC がコンテキストに加えてその証明書と CtxC-id の対応を CAP に送信できる場合について、管理者を暗黙的に信頼することなくコンテキストを紐付ける方法を提案する。証明書を用いた公開鍵認証は、ユーザーやデバイスの認証手段の 1 つである。特に、ハードウェアセキュリティモジュール内に証明書の秘密鍵を格納しておくことは、ZTA における有効な認証手段でもある [5]。

本提案で利用可能な証明書の例として、X.509 証明書 [2] であったり、入退室を管理するような IC カードに格納されている証明書が挙げられる。

この場合では、エンティティが CtxC で認証に利用している証明書の秘密鍵の所有が CAP で確かめられれば良い。具体的な方法として、図 5 と図 6 を用いて説明する。また、CtxC はコンテキストに証明書の一部分を識別子として含めて CAP に送信しているものとする。

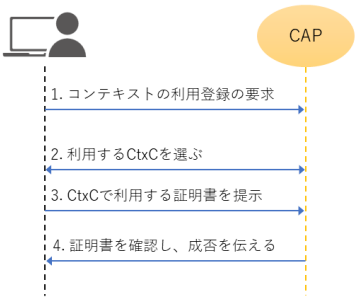

図 6　コンテキストの利用登録フロー

次に、コンテキストを利用可能な状態にするコンテキストの利用登録フローを図 6 を用いて説明する。まず、エンティティは CAP に利用登録の申請を行う。これに対して CAP はエンティティに、利用可能な CtxC の内、利用登録を行う CtxC を確認する。エンティティは利用登録したい CtxC を選び、CtxC で利用している証明書を電子署名によって提示する。CAP で署名検証を行い、証明書からシリアルなどの認証に必要な情報を抜き出す。この結果、認証に成功していれば利用登録の成否をエンティティに伝える。この一連の手続きにより CAP はエンティティを CtxC-id の正当な保有者として登録し、エンティティとコンテキストを紐付けることができる。

この設計においても、3.4.1 節と同様に CtxC における拡張実装はほとんど必要ではなく、CAP にコンテキストを送信する部分についてのみ実装すればよい。

ここでは簡単のために CtxC から CAP に証明書を CtxC-id としてコンテキストを送るという説明を行ったが、証明書と CtxC-id の対を何らかの方法で送る場合も考えられる。また、本章では CtxC から CAP にコンテキストを送るという説明を行ったが、CtxC で API を用意しておいて、CAP がそこから取得するという場合も考えられる。伝送する方法については、なりすましやコンテキストに含まれる個人情報の流出を防ぐため、mutual-TLS[9] のようになりすましができないようにしつつ暗号化することが望ましいが、具体的な方法は本研究の範囲外である。

4.　CtxC と CAP の実装例

4.1　シナリオ

本章では、デバイスのコンテキストを追加実装が困難な CtxC からコンテキストを収集する実装として、802.1X[1] の RADIUS サーバーと Microsoft の MDM サービスである Intune[*1] を例に、3.4.2 節で示した CtxC から受信した証明書を含むコンテキストを CAP が紐付ける方法を示す。

[*1] https://docs.microsoft.com/ja-jp/mem/intune/fundamentals/what-is-intune

このシナリオでは RADIUS サーバーや Intune を CtxC として、RP が CAP から RADIUS サーバーで得られる通信ログと Intune で得られるデバイスの情報をコンテキストとして受け取り、認証認可を行うことを考える。RADIUS サーバーや Intune を CtxC として利用することで、現在接続しているのか、どのアクセスポイントや LAN スイッチに接続しているのかといったデバイスの位置を調べるのに有用な情報や、OS バージョンなどの脆弱性が知られていないデバイスを利用しているのかといったデバイスの安全性に関する情報を得ることができる。

これらのコンテキストをアクセス制御に用いることで、RP はより細やかな認可判断を行うことができる。例えば、コワーキングスペースがこの CAP を提供しており、部屋ごとに無線 LAN アクセスポイントがあることを仮定すると、デバイスが現在どのような性質を持った部屋にあるのかを知ることができる。覗き見られることのない鍵付きの個室にいるかどうかを知ることができれば、RP は機密情報の閲覧を認可するための判断材料の 1 つにできると考えらえれる。また、完全防音の部屋にいる時には RP は機密情報が含まれる音声や動画のストリーミングを認可するための判断材料にできると考えられる。

この実装では、3.4.2 節にて提案した方法を利用する。RADIUS サーバーについて、802.1X で利用する認証プロトコルに EAP-TLS[11] のように証明書を用いるプロトコルを利用するものとし、Intune については Intune の機能でデバイスの証明書を管理している状況を考える。これにより、3.4.2 節での仮定を満たす。それぞれから CAP にコンテキストと CtxC-id と証明書の対応を送信し、CAP でデバイスに証明書で認証させることで紐付けを行う。

4.2 概要

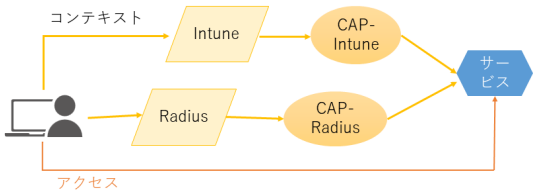

図 7 概要図

図 7 に本実装の概要を示す。RADIUS サーバーと Intune は CtxC となり、それぞれの CAP にコンテキストとしてデバイスの通信ログやデバイスの状態と CtxC-id と証明書の対応を定期的に送信する。その後、CAP は CtxC から収集したコンテキストをデバイスと紐付けるために、CtxC から得た証明書をデバイスに提示させる。提示する際には証明書に対応する秘密鍵を用いて、デバイスが所有する証明書であることを署名を用いて証明する。CAP は提示された署名を検証し、確認できた証明書と CAP-id を紐付けることで、コンテキストとデバイスを CAP 内で対応させる。この一連の手続きを行うことで CtxC から取得したコンテキストを RP に提供できるようになる。

4.3 デバイスの通信ログの送信 (CtxC → CAP)

送信の際には安全かつ相手になりすましされないような方法でデバイスの情報を送る必要がある。コンテキストには重要な個人情報が含まれる可能性があり、そのようなコンテキストが攻撃者によって盗まれるようなことがないようにするためである。また、攻撃者が CtxC に偽装して攻撃者に有利なコンテキストを CAP に注入されることがないように、正しい CtxC からコンテキストが送られてきていることを確認する必要もある。そのため、mutual TLS(mTLS) のような相互になりすましができず、暗号化される方法で送ることが求められる。

4.4 デバイスの CAP に対する証明書の提示

CAP-id とコンテキストを紐付けるために、デバイスが CtxC で利用している証明書を CAP に提示する。提示の際には、証明書に対応する秘密鍵を用いて署名検証を行うことで証明書を所有していることを証明する。また、CtxC で利用されることがないデバイスが証明書を提示している可能性を排除するために、CtxC で証明書の検証に使われている上位の認証局の証明書を利用して証明書を検証する。

4.5 コンテキストとデバイスの紐付け

4.3 節で CAP はデバイスが RADIUS サーバーに提示する証明書を得る手法を示した。コンテキストにはこの証明書に関する情報が含まれているため、デバイスが直接提示した証明書と比較して、同一の証明書だと判断できるデバイスを CtxC-id に対応するとして紐付ける。証明書の発行元とシリアル番号が一致するものを同一の証明書とみなす。

4.6 実装環境 (Radius)

今回の実装では、FreeRADIUS[*2]のサーバーを CtxC として利用した。また、802.1X の認証装置として研究室の無線 LAN アクセスポイントを利用した。この無線 LAN アクセスポイントは EAP-TLS による認証を行っており、認証サーバーとして当該 RADIUS サーバーを利用している。実装言語は、CAP の実装においては Go 言語[*3]と Go 言語で利用される Web フレームワークである Echo[*4]を利用し、コンテキストを CtxC から CAP に送信する部分は Bash[*5]で実装した。また、CAP は HTTPS サーバーであるため、実装に際して Open SSL[*6]を用いて自己署名証明

*2 https://freeradius.org/
*3 https://go.dev/
*4 https://github.com/labstack/echo
*5 https://www.gnu.org/software/bash/
*6 https://www.openssl.org/

書を発行している。

RADIUS サーバーはコンテキストをファイルに保存しているため、そのファイルを監視し、cron で 5 分ごとに変更があるか確認し、変更があればシェルスクリプトなどて差分を CAP に送信するようにした。監視するファイルは二種類あり、1 つが RADIUS の認証ログであり、もう 1 つはアカウンティングログである。

コンテキストとして送信している認証ログのフィールドの内、主要なものは以下の通りである。

TLS-Client-Cert-Serial EAP-TLS で提示されたクライアント証明書のシリアル番号

TLS-Client-Cert-Issuer クライアント証明書の発行元

Calling-Station-Id 送信元の MAC アドレス。アカウンティングログには証明書の情報は含まれないため、認証後のセッション中はこれを用いて識別する。

Timestamp Unix 時間のタイムスタンプ

これらのログを用いることで、CAP はデバイスを認証し、コンテキストを紐付けることができる。

コンテキストとして送信しているアカウンティングログのフィールドの内、主要なものは以下の通りである。

Calling-Station-Id 送信元の MAC アドレス。これでデバイスを識別する。

Acct-Status-Type Start, Interim-Update, Stop のいずれかの値をとり、それぞれ接続開始、接続している間の定時報告、接続終了を意味する。

Acct-Session-Id 接続開始から接続終了まで保たれるセッション識別子。デバイスの識別の際にこれが等しいことを確認している。

Acct-Input-Octets デバイスが受信した通信量 (byte)

Acct-Output-Octets デバイスがで送信した通信量 (byte)

Timestamp Unix 時間のタイムスタンプ

CAP はこれらのコンテキストを受け取った際に、コンテキストを RP などに提供しやすい形にまとめておく。実装では認証ログの Calling-Station-Id を用いて同一のデバイスのコンテキストをまとめ、現在接続中であるかについての bool 値や最後に接続した時刻のタイムスタンプを持つなど、RP に提供しやすい形に加工し保管した。

RADIUS サーバーで認証に利用している証明書をデバイスに提示させる部分の実装には Go 言語標準のライブラリを用いて、mTLS によって提示させ、証明書の署名を検証した。これにより、CAP はコンテキストを紐付けることことができる。

4.7 実装環境 (Intune)

Microsoft Intune の Managed Device の API[*7] に CAP から定期的にアクセスすることでコンテキストを得た。コンテキストの紐付けに使う証明書は本来 Intune で管理させるべきであるが、今回は notes フィールドに証明書の情報を直接書き込んでおくことで実装を簡略化した。

今回コンテキストとして利用したフィールドは以下の通りである。

id デバイスの ID(CtxC-id に相当)

operatingSystem デバイスのオペレーティングシステム

osVersion デバイスの OS バージョン

managementState デバイスの管理状態

isEncrypted 暗号化されているか否か

lastSyncDateTime 最後に Intune と同期した時刻

notes 自由に扱えるフィールド

OS バージョンは脆弱性が知られていないバージョンのデバイスであるのかを知るために使うことができる。また、デバイスの管理状態や暗号化されているかや Intune との最終同期時刻からデバイスが適切に管理されている状態なのかを知ることができる。これらの情報をコンテキストとして用いて、ZTF の実装を行った。

5. 考察

本研究では、CtxC から CAP へのコンテキスト送信時のコンテキストの紐付けについて議論した。また、CtxC での拡張実装が難しい場合として管理者による手動での紐付けと証明書を用いた検証による紐付けの 2 種類の方法を提案した。

ここで、証明書での検証の安全性について考えるために、入退室に用いるカードキーの使用履歴からユーザーの居場所についてコンテキストを与える CtxC について同一の管理者を仮定せずに考える。カードキーの中には証明書が入っており、その証明書で認証が行われていると仮定する。また CtxC として入退室ログとその際の証明書の情報を保存するデータベースがあり、それを CAP に送信する場合について考える。このカードキー内の証明書の情報を用いてコンテキストを紐付けたい。これを行うためには、デバイスの NFC リーダーなどでカードキーから証明書の情報を読み取り、CAP に証明書を提示すればよい。しかし、この方式ではカードキーが攻撃者にスキミングされた際に攻撃者が居場所をユーザーと同じ場所になりすませる危険がある。この対策として、証明書にパスフレーズを設定し、デバイスの NFC リーダーで読み込んでもパスフレーズがなければ認証ができないようにするのは効果的な対策である。証明書を利用するとパスフレーズが盗まれた場合でも証明書を無効化することで被害を最小化できるという利点もある。この証明書の無効化を行う際には、古い行動履歴

[*7] https://docs.microsoft.com/ja-jp/graph/api/resources/intune-devices-manageddevice?view=graph-rest-beta

が取れなくなってしまうことを防ぐために新しい証明書と古い証明書を対応させるなどの処理が必要になると考えられる。この場合には認証に用いる証明書に、証明書に依存しないユーザーの識別子を事前に含めておき、CAP ではその識別子を用いて検証を行うように実装しておくことで、前の証明書を用いていた時の行動履歴が失われることを防ぐことができる。

証明書のような検証できる識別子を用いることなく同様の手法でコンテキストを紐付けるべきではないことを確認するために、前述のカードキーの例の仮定を一部変更し、FeliCa の IDm のような検証不可能な識別子のみによって入退室を管理するようなケースを考える[*8]。このような場合においても、証明書を識別子とする場合と同様に考えると、IDm をデバイスの NFC リーダーで読み取り、それを CAP に提示するという方法が考えられる。しかし、この方法ではその IDm を持つカードを利用していることを CAP が検証できないため、攻撃者が偽装クライアントから不正な IDm を送ることできてしまう。どのような方法であっても IDm をデバイスから直接 CAP に送る必要があるという点は避けられないず、IDm を直接送る際には偽装クライアントを用いた攻撃を回避するのは困難であるため、CAP は何らかの手段で検証できる識別子しか受け入れるべきではない。

6. 結論

IdF 下で ZTA における認証認可を行うためのゼロトラスト認証認可連携 (Zero Trust Federation; ZTF)[4] という枠組みの実用化を目指し、本研究では ZTF においてコンテキストを収集し提供する役割を持つ CAP が、RP 以外からコンテキストを収集する手法について提案した。コンテキストを直接収集して CAP に与える Context Collector(CtxC) というエンティティを定義して、CAP の役割を整理した。これにより、CtxC から CAP がコンテキストを得るための課題として、CtxC における識別子 (CtxC-id) と CAP における識別子 (CAP-id) の対応を CAP が得ることが必要であることを明確にした。これを一切の仮定を置かずに達成することは困難であるので、追加実装が容易な場合、CtxC と CAP の管理者が同一の場合、CtxC で証明書による認証を行う場合の 3 つの場合に分けてこの問題に取り組んだ。更に、RADIUS サーバーが CtxC である場合について提案手法を実装し、証明書を CtxC が認証に利用している場合のコンテキストの紐付けの方法を具体的に説明した。紐付けることができることを確認した。

今回の提案手法によって利用できるコンテキストは、CtxC が特定の条件を持ったケースについてのみである。より多様なコンテキストを利用できるようにすることは、より堅牢にリソースを守るために細やかな認可判断を行う上で不可欠である。よって今後の課題として、より広範な条件を持った CtxC のコンテキストを RP が利用できるように、更なる手法の考案が望まれる。

[*8] FeliCa の IDm は偽造が容易であるため、そもそも認証を行うのは適さないが、これを使って入退室管理を行っている例は少なくないため、偽造が不可能であるという仮定のもとでこの場合を取り扱う。

参考文献

[1] IEEE Standard for Local and Metropolitan Area Networks–Port-Based Network Access Control, *IEEE Std 802.1X-2020 (Revision of IEEE Std 802.1X-2010 Incorporating IEEE Std 802.1Xbx-2014 and IEEE Std 802.1Xck-2018)*, pp. 1–289 (online), DOI: 10.1109/IEEESTD.2020.9018454 (2020).

[2] Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S. and Cooper, D.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, RFC 5280 (2008).

[3] Hardt, D.: The OAuth 2.0 Authorization Framework, RFC 6749 (2012).

[4] Hatakeyama, K., Kotani, D. and Okabe, Y.: Zero Trust Federation: Sharing Context under User Control towards Zero Trust in Identity Federation, *2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 514–519 (online), DOI: 10.1109/PerComWorkshops51409.2021.9431116 (2021).

[5] Kindervag, J. et al.: Build security into your network's dna: The zero trust network architecture, *Forrester Research Inc*, Vol. 27 (2010).

[6] Maciej Machulak, J. R.: Federated Authorization for User-Managed Access (UMA) 2.0, `https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-federated-authz-2.0.html` (2018). (Accessed on 02/01/2022).

[7] OASIS Security Services TC: Security Assertion Markup Language (SAML) V2.0 Technical Overview, `https://www.oasis-open.org/committees/download.php/27819/sstc-saml-tech-overview-2.0-cd-02.pdf` (2008).

[8] OpenID Foundation: Final: OpenID Connect Core 1.0 incorporating errata set 1, `https://openid.net/specs/openid-connect-core-1_0.html` (2014). (Accessed on 01/27/2022).

[9] Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3, RFC 8446 (2018).

[10] Rose, S., Borchert, O., Mitchell, S. and Connelly, S.: Zero Trust Architecture, NIST SP 800-207 (2019).

[11] Simon, D., Hurst, R. and Aboba, D. B. D.: The EAP-TLS Authentication Protocol, RFC 5216 (2008).

[12] Tulshibagwale, A.: Re-thinking federated identity with the Continuous Access Evaluation Protocol, https://cloud.google.com/blog/products/identity-security/re-thinking-federated-identity-with-the-continuous-access-evaluation-protocol.