Linking Contexts from Distinct Data Sources in Zero Trust Federation

Masato Hirai, Daisuke Kotani, and Yasuo Okabe

Kyoto University, Yoshida-honmachi, Sakyo-ku, Kyoto 606-8501 JAPAN

Abstract. An access control model called Zero Trust Architecture (ZTA) has attracted attention. ZTA uses information of users and devices, called context, to verify access requests. Zero Trust Federation (ZTF) has been proposed as a framework for extending an idea of identity federation to support ZTA. ZTF defines CAP as the entity that collects context and provides it to each organization (Relying Party; RP) that needs context for verification based on ZTA. For precise verification, CAPs need to collect context from various data sources. However, ZTF did not provide a method for collecting context from data sources other than RP. In this research, as a general model for collecting context in ZTF, we propose a method of linking identifiers between the data source and CAP. This method provides a way to collect context from some of such data sources in ZTF. Then, we implemented our method using RADIUS and MDM as data sources and confirmed that their contexts could be collected and used.

Keywords: Access Control · Context · Zero Trust.

1 Introduction

In Zero Trust Architecture (ZTA)[7], an organization always verifies the origin of access using data called context and authorizes access using the verification results. Context includes continuously changing information such as the location of the user or device, access history, as well as the surrounding conditions of the accessing source. Context should be collected from various data sources for precise verification.

To extend ZTA for Identity Federation(IdF), Zero Trust Federation (ZTF) [2] is proposed as a model to extend ZTA to make authorization decisions using the context of multiple organizations in IdF. The ZTF introduces Context Attribute Provider (CAP) as an entity to share context in the federation. CAP collects context independently of the organizations and provides context to each organization (Relying Party; RP) with the user's authorization. However, ZTF did not provide the method of CAPs collecting context from sources other than RPs. To make more precise authorization decisions in ZTF, it needs to collect diverse contexts from more data sources, including those other than RPs.

The data sources that can provide context are diverse. For the data source to work as CAP, a huge amount of additional implementations are required,

including communication with an authorization server to obtain the user's authorization status and sending the context to the RP. These implementations are not always possible for some data sources. Designing for each case would increase the cost of implementation, making it difficult to collect context from various data sources.

This study discusses the methods of collecting context to show the desirable architecture as a CAP, and proposes design for data sources that are difficult to make additional implementaion, which requires particular consideration.

The relationship between data source and CAP was unclear although CAP was proposed to collect and provide context in [2]. Therefore, we transfer CAP's role of collecting context to data sources and define the data sources as Context Collector (CtxC). This leaves the CAP's role only to provide contexts, and what we should discuss is how to send context from CtxC to the CAP. Contexts collected by CtxC usually contain the CtxC's unique user or device identifier (CtxC-id). For the CAP to provide the context received from CtxC to each RP, the CAP must determine to which user or device in CAP the context should be mapped. Thus, the problem is how to map the CtxC-id in the context to an identifer in the CAP(CAP-id). This study refers to this as linking context.

In this study, we propose a design for CtxC to perform the linking context in three cases: (1) a case where CtxC is easy to be extended for pseudonymous ID sharing with CAP, which was proposed in the previous ZTF; (2) a case where the administrator of CtxC and CAP is the same, which does not require additional implementation but trusts the administrators of both CtxC and CAP; (3) a case where CtxC uses certificates to authenticate devices or users, which does not require to trust the administrator links contexts properly. In (1), the CtxC and CAP share a pseudonymous ID for each user or device in advance, and the CtxC includes the pseudonymous ID in the context before sending it to the CAP so that the CAP can link the context to CAP-id. In (2), the administrator confirms the correspondence between identifiers in CtxC and CAP directly and places the correspondence table of the identifiers in CAP. In (3), CAP requests the certificate used in CtxC from the user or device and links identifiers which are related to the same certificates.

Furthermore, as a specific design for case (3), this study presents implementations for authentication and authorization by verifying factors such as which LAN the device is connected to and whether the device's OS has been compromised. As in CtxC, one implementation uses a RADIUS server using EAP-TLS and the other uses MDM. Through these implementations, we show the way of linking contexts using certificates.

2 Related Research

ZTA[7] is a new access control model in contrast to the traditional access control method, the perimeter model. ZTA controls access by continuously verifying access requests. To verify access, ZTA uses context, which includes static infor-

mation as well as dynamic information such as the situation surrounding the user or device.

ZTF[2] is a framework for IdFs to federate the context to authenticate and authorize users like ZTA. ZTA typically has a single organization centrally collecting context[3]. However, in IdFs, contexts are dispersed across multiple RPs and cannot be aggregated. Therefore, ZTF defines Context Attribute Provider (CAP), which collects and provides context across IdFs. CAP is proposed as an entity that enables authentication and authorization using sufficient types and amounts of context, even for IdFs accessed infrequently.

As a method for CAP to provide context to the RP, ZTF proposes using CAEP[9] for continuous authentication and authorization and UMA[4] to authorize access under user's authorization.

However, it was unclear in the ZTF how to collect context from sources other than RPs. Collecting various contexts from sources including non-RP will improve authorization quality and will be required. For example, by using records of entering and leaving a room as context, it is possible to know where users are. On the other hand, the system of collecting such records does not provide a way of authenticating users directly via network. If the system is designed with different authentication requirements, it is too difficult to implement such authentication features. Therefore, the same protocols cannot be used as RPs when sharing a context with the RP.

3 The Method of Linking Context

3.1 Definition of Context Collector(CtxC)

Context is collected by RPs and CAPs in ZTF, but there are important data sources that belong neither to RP nor CAP. In order for such data source to be a CAP, the data source must manage the context based on the user's authorization to provide the context to the RP. However, these features are not implementable for all data sources, such as embedded systems. Therefore, to pursue a more desirable architecture, we define a data source as a Context Collector (CtxC). This means that we separate CAP's roles into collection and provision of context, and transfer the collection role to CtxC. In other words, CtxC is responsible for collecting the context, and CAP is responsible for verifying the user's authorization and providing the context to the RP. We redefine the CAP as always collecting context indirectly from the CtxC.

This change in ZTF is illustrated in Fig. 1. CAP2 receives the context indirectly from CtxC1. Also, RP2 is regarded as CtxC because it collects context directly from the user. A single CAP handles one or more CtxCs.

The Reason Why Linking Context Is Necessary

When CtxC collects contexts, a user or a device that the contexts express is usually identified with CtxC's user/device identifier (CtxC-id). In contrast to CtxC-id, we call the identifier of the user in CAP as CAP-id. CAP must check the mapping between CtxC-id and CAP-id to manage user's authorization of RPs.

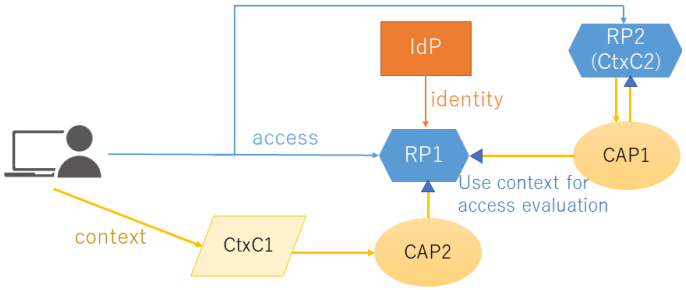

Fig. 1. ZTF organized by defining CtxC

Once CAP obtains that mapping, it can manage authorization status using such protocols as OAuth2.0 and UMA. Thus, to provide the context for RPs properly, CAP needs a method of mapping between the CtxC-id and the CAP-id. We call this mapping as linking context, and discuss the method of linking context. Since it is difficult to link context without making any assumptions about CtxC or CtxC-id, we considered three cases based on real use cases. We propose solutions for each of the cases below. One is the case where CtxC is easily extensible to share IDs using some protocols as in web applications. Another is the case where CtxC and the CAP administrator are the same. This case will be applicable for the entry/exit record system in a company. The other is the case where CtxC authenticates with client certificates. This case can apply to Radius server using EAP-TLS[8].

3.2 Linking context

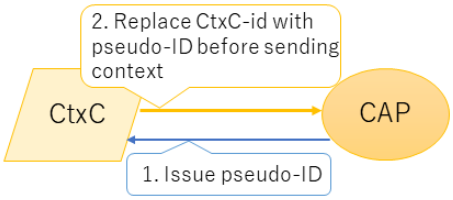

Fig. 2. The case where CtxC is easily extensible

When CtxC is easily extensible In this case, we assume CtxC is so extensible that it can share pseudonymous ID. This is applicable for web applications. OpenID Connect(OIDC)[6] and SAML[5] are known as protocols for use of pseudonymous IDs.

The way of linking contexts is explained in Fig. 2. First, CAP issues a pseudonymous identifier (pseudo-ID) to CtxC. At this time, CAP stores the correspondence between the pseudo-ID and a CAP-id, and CtxC stores the correspondence between the pseudo-ID and a CtxC-id. The issue of pseudo-IDs can be implemented, for example, using OpenID Connect ID tokens [6]. CtxC then replaces the CtxC-id in the context with the pseudo-ID and sends it to CAP. This procedure allows the CAP to receive contexts with pseudo-IDs and to link

contexts easily. Also, in this procedure, CtxC and CAP are exchangeable so that CtxC may issue pseudo-ID.

When CtxC is not easily extensible In this and subsequent sections, we will discuss methods of linking contexts in consideration of cases where CtxC is not easily extensible. In this case, we assume that no additional implementation to current implementation, such as embedded systems, can be made. For CtxC and CAP to share a pseudo-ID, CtxC must authenticate the user via a browser or native application to map the pseudo-ID and CtxC-id. Doing this would require additional implementation on CtxC's authentication for users or devices. However, this additional implementation is not realistic for this case.

Therefore, in this study, we have designed the CtxC to implement additionally only a feature of transmitting context to CAP, which is independent of the existing CtxC's implementation.

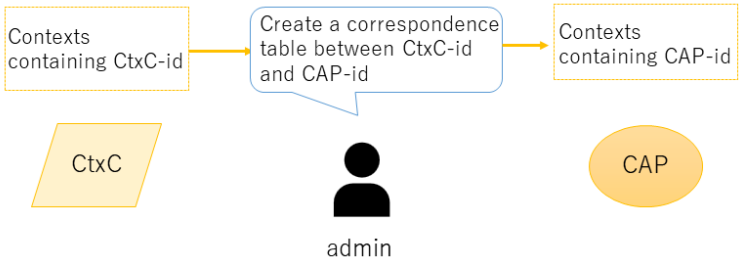

Fig. 3. Administrator associates the CtxC-id with the CAP-id.

When CtxC and CAP have the same administrator As shown in Fig. 3, in this case, the administrator knows the correspondence between a CtxC-id and a CAP-id so they can create the correspondence table in the CAP. CAP uses this table to link contexts.

For example, suppose that a company operates CAP and that the company would manage an entrance control system using IC cards as CtxC. When registering an IC card at the time of joining the company, the administrator creates a correspondence between the IC card identifier and the employee ID and registers it in the CAP, so that the CAP can easily link the context.

With this approach, it is only necessary to transmit the context from CtxC to CAP in some way, and little additional implementation is required. However, in this method, the CAP must trust that the administrator will maintain the correspondence table constantly.

When CtxC authenticates using certificates The method above requires implicit trust that the administrator will not link wrong context. Therefore, we propose a method of linking contexts without implicitly trusting the administrator. In this case, we assume the CtxC uses certificates for authentication and can send the correspondence between a certificate and CtxC-id to CAP with context. We also assume that the CA issues certificates correctly. Examples of certificates available in this proposal are X.509 certificates and certificates stored in IC cards that control entering/exiting rooms.

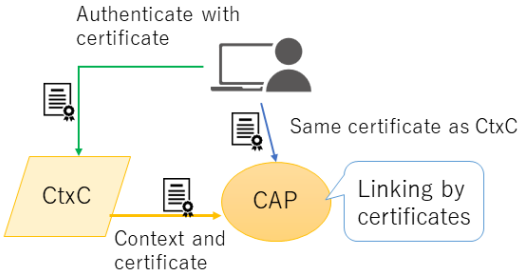

Fig. 4. Overview of authentication by certificate

In this assumption, CAP should verify the user/device has the private key of the certificate that the user/device uses for authentication in CtxC. The method is illustrated in Fig. 4.

We provide an overview in Fig. 4. As shown by the green arrow, CtxC requests a certificate from the user/device and authenticates using it. Then, CtxC sends the certificate and context to CAP. The CAP then requests the same certificate from the user/device used in CtxC to verify that the user/device has the certificate's private key. It also verifies that the certificate has not been modified by validating the certificate chain. This procedure allows the CAP to link contexts corresponding to certificates.

In this design, little additional implementation in CtxC is required. Only the feature of sending contexts to the CAP is needed. Also, we explained that the context is sent from CtxC to CAP, our method is applicable for a case where CtxC prepares an API and CAP obtains the context from it.

4 An Example of CtxC and CAP implementation

As example implementations to collect context from CtxC, this section shows how CAP links context with the certificates, using 802.1X[1] RADIUS server and a MDM service. This implementation is published on GitHub[1].

In this scenario, we consider the RADIUS server and MDM as CtxC. The RADIUS server can collect connection logs in LAN and MDM can collect devices' states. The connection logs have the information such as which access point the device connects, and are useful to locate the device. Also, the devices' states have the information such as whether the device's OS has no known vulnerabilities. The RP can use these pieces of information as context to control access precisely.

We implemented the method of using certificates, which is explained in section 3. In our implementation, the Radius server authenticates devices using 802.1X EAP-TLS[8] and the MDM manages device certificates. These satisfy the assumptions as CtxCs in that method. Each CtxC sends a correspondence between context and certificate to the CAP, and the CAP links contexts by the certificate.

Fig. 5 shows an overview of our implementation. As CtxCs, the Radius server sends connection logs to CAP-Radius and the MDM sends devices' states to

[1] https://github.com/laft2/cap-demo

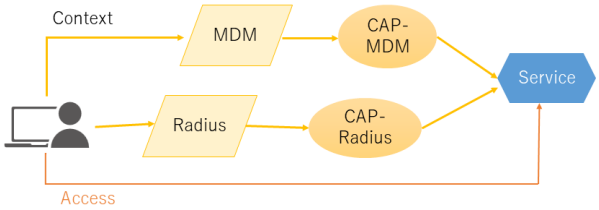

Fig. 5. Overview of implement

CAP-MDM. They also send certificates with the context to CAPs. The CAP then requests the device with certificate used in CtxC to link the context. Afterward, the CAP associates the certificate with the CAP-id of the device. These procedures allow the CAP to provide the RP with the context obtained from CtxC.

In this implementation, we used the FreeRADIUS server in our laboratory as CtxC. Also, we used a Wi-Fi access point in the laboratory as an 802.1X authenticator. The Radius server controls access for the Wi-Fi access point to use EAP-TLS as an authentication protocol. For the CAP-Radius implementation, we used the Go and Echo, a web framework for Go.

We monitored and sent two files, one is the RADIUS authentication log, and the other is the accounting log. We have used fields of TLS-Client-Cert-Serial and TLS-Client-Cert-Issuer in the authentication logs. The CAP-Radius can authenticate the device and link the context using these pieces of information. From the accounting logs, we have used the field of Acct-Status-Type, which has the change of device's connectivity status as context. We also have used the fields of Acct-Input-Octets and Acct-Output-Octets, which express the device's traffic. We implemented CAP-Radius to turn these contexts into useful states such as whether the device is connected now.

We designed the CAP-Intune to gain context by periodically accessing Microsoft Intune's Managed Device API[2]. Intune is Microsoft's MDM service. We have used the fields of osVersion, complianceState, lostModeState, and jailBroken to confirm that the device is securely maintained. For example, we can use the OS version to know if the device is known as a non-compromised OS. We implemented the ZTF using this information as a context.

5 Concluding Remarks

Zero Trust Federation (ZTF)[2] is a framework for authentication and authorization in ZTA under IdF. To clarify the relationship between CAP and the data source, we separate CAP's roles into the collection and provision of context, and transfer the collection role to other entity we define as CtxC. This separate leaves the CAP's role only to provide context. We clarifies that the problem for CAP to obtain the context from CtxC is that the CAP must obtain a correspondence between the CtxC-id and CAP-id. It is difficult to achieve

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

this without making any assumptions. Therefore, we addressed this problem in three cases based on use cases: when additional implementation is easy like web applications, when the administrators of the CtxC and the CAP are the same like recorders of enter/exit the rooms, and when the CtxC authenticates with a certificate like Radius servers using EAP-TLS. Furthermore, we implemented the proposed method for the case where the RADIUS server and Intune are CtxC. This implementation specifically shows the method of linking contexts when CtxC uses certificates for authentication.

The only context available by the proposed method is for cases where CtxC satisfies certain assumptions. The availability of more diverse contexts is essential for making precise authorization decisions to more robustly protect resources. Therefore, in future work, further methods should be devised to allow RPs to use CtxC contexts with a wider range of conditions.

References

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 (2020). https://doi.org/10.1109/IEEESTD.2020.9018454
2. Hatakeyama, K., Kotani, D., Okabe, Y.: Zero trust federation: Sharing context under user control towards zero trust in identity federation. In: 2021 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops). pp. 514–519 (2021). https://doi.org/10.1109/PerComWorkshops51409.2021.9431116
3. Kindervag, J., et al.: Build security into your network's dna: The zero trust network architecture. Forrester Research Inc **27** (2010)
4. 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` (1 2018), (Accessed on 02/01/2022)
5. 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` (3 2008)
6. OpenID Foundation: Final: Openid connect core 1.0 incorporating errata set 1. `https://openid.net/specs/openid-connect-core-1_0.html` (11 2014), (Accessed on 01/27/2022)
7. Rose, S., Borchert, O., Mitchell, S., Connelly, S.: Zero trust architecture. NIST SP 800-207 (9 2019), `\url{https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-207-draft.pdf}`
8. Simon, D., Hurst, R., Aboba, D.B.D.: The EAP-TLS Authentication Protocol. RFC 5216 (Mar 2008). https://doi.org/10.17487/RFC5216, `https://www.rfc-editor.org/info/rfc5216`
9. 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 (2 2019)