A Structured Security and Privacy Assessment of Unity Authentication for Immersive Applications

Mihnea-Vlad NICOLAE, Ionuț-Andrei MOCANU, Răzvan-Alexandru DUȚESCU, Cătălin-Andrei NICULESCU
National Institute for Research & Development in Informatics - ICI Bucharest
mihnea.nicolae@ici.ro, ionut.mocanu@ici.ro, razvan.dutescu@ici.ro, catalin.niculescu@ici.ro

Abstract: This paper presents a structured security and privacy assessment of Unity Authentication approaches for immersive applications, with emphasis on Unity Player Accounts, OpenID Connect, and Custom ID sign-in. The study combines documentation analysis, prototype implementation, and Wireshark-based traffic observation to identify developer responsibilities and practical integration risks. The analysis focuses on trust boundaries, token handling, account linking, session persistence, backend hardening, service-account protection, and GDPR-relevant data processing. A Unity XR prototype was implemented to examine the Unity Player Accounts sign-in workflow and to identify authentication-related network endpoints during passive Wireshark observation. The results show that HTTPS/TLS protects authentication payload confidentiality during passive capture, but observable metadata still reveals infrastructure dependencies and identity-provider involvement. The paper argues that the main security risks in Unity-based authentication are implementation-level weaknesses rather than transport-layer exposure. The contribution is a reproducible assessment framework and a set of practical recommendations for integrating Unity Authentication securely and privacy-consciously in metaverse and XR environments. Unlike prior work that addresses metaverse authentication at a general level, this paper focuses specifically on Unity Authentication as a commercial game-engine identity layer and evaluates its integration risks in an XR prototype.

Keywords: Identity management, User authentication, Unity Gaming Services, Service account protection, Custom ID sign-in

INTRODUCTION

Modern applications and games rely on robust identity management systems to authenticate users, enable cross-platform connectivity, and provide secure access to backend services. Unity Authentication, part of Unity Gaming Services, offers a variety of sign-in methods, including anonymous login, external identity providers via OpenID Connect (OIDC), and

https://doi.org/10.54851/v8i1y202602

Custom ID sign-in. Each approach has its distinct security and privacy implications: OIDC allows integration with established identity providers, while Custom ID uses a server-authoritative model for authentication. Understanding the responsibilities of the developer over Unity itself, as well as the risks associated with token exposure, account linking, and privileged credential management, is crucial for building secure and privacy-compliant applications and games.

From a privacy perspective, authentication identifiers are considered personal data, particularly when linked to external identities or persistent session tokens. Unity Authentication supports GDPR-compliant processing, but the accountability for data protection lies in the hands of the developer, who determines how identifiers are collected, stored, and shared. Security considerations in client-server architectures are equally important: client environments are inherently public and vulnerable to token leakage, while backends must protect accounts, validate inputs, and implement rate-limiting and auditing. By addressing these issues proactively, combining hardened server endpoints, minimal data collection, and transparent user data workflows, developers can ensure that Unity Authentication is both a secure and privacy-respecting foundation for games and metaverse environments.

This paper makes three contributions. First, it provides a threat-model-based comparison of Unity Authentication approaches, focusing on Unity Player Accounts, OIDC-based sign-in, and Custom ID sign-in. Second, it analyzes the privacy implications of Unity Authentication under GDPR-oriented controller/processor role allocation, with emphasis on persistent identifiers, external identity claims, and account-linking behavior. Third, it presents an XR implementation case study using Unity Player Accounts and interprets Wireshark-based traffic observations in terms of observable metadata, infrastructure dependencies, and the limitations of passive encrypted-traffic analysis.

RELATED WORK AND POSITIONING OF THE PRESENT STUDY

Prior research on authentication in metaverse and XR environments has mainly focused on general identity risks, biometric authentication, multi-factor authentication, decentralized identity, and privacy-preserving identity mechanisms. These studies are valuable because they identify broad authentication challenges in immersive environments, including identity theft, biometric-data exposure, user tracking, and the need for stronger access control. However, they generally do not examine how commercial game-engine authentication services are integrated in practice, nor do they provide a Unity-specific comparison of Player Accounts, OpenID Connect, and Custom ID sign-in. The present paper addresses this narrower implementation gap by analyzing Unity Authentication from the perspective of trust boundaries, developer responsibility, token handling, GDPR-relevant identifiers, and observable network communication during an XR sign-in workflow.

Security in the metaverse is a wide subject that has been studied in depth, and a substantial body of literature has addressed this topic. There are numerous approaches to doing metaverse security. One of the most popular approaches is multifactor authentication (MFA). This method combines multiple categories of proof, typically something the user knows, such as a password, something the user has such as a tokeniser on a trusted device and something the user is such as a biometric trait. MFA is promising because it improves resilience against spoofing and other single-point failures (Hallal, Rhinelander & Venkat, 2024). A more advanced MFA authentication method for the Metaverse is called adaptive MFA. This method adjusts authentication requirements based on context, such as a low-risk environment might only require platform login, but more protected simulations, records, or accessing virtual assets will require additional biometric or behavioral steps (Kang, Park & Kim, 2025). In relation to the present study, MFA and adaptive MFA are

relevant mainly as upstream identity-provider capabilities. Unity Player Accounts or OIDC-based sign-in may rely on an external provider that implements MFA, but Unity Authentication itself remains responsible for mapping the authenticated identity to a Unity player identity and maintaining the player session.

Another widely studied authentication method is biometric authentication. This method refers to the use of measurable human characteristics in order to verify the identity of the user. This includes but is not limited to face detection and recognition, iris measurements, fingerprint matching, or voice verification (Hallal, Rhinelander & Venkat, 2024). This method of authentication is low-friction when compared to traditional login methods. This means that the user will log in using a device embedded in a controller or on their phone, rather than having to enter their password or wait for and receive a one-time code (Kang, Park & Kim, 2025). There are, however, several privacy-related drawbacks and challenges with this method. For Unity-based immersive applications, biometric authentication is therefore best understood as a possible identity provider or platform-level mechanism rather than a direct replacement for Unity Authentication. The main Unity-specific issue remains how the result of biometric or platform authentication is translated into a secure player account without excessive data collection or unsafe token handling.

An authentication method that preserves the privacy of the user is decentralized identity. (Kim et al., 2023). This authentication model involves the user controlling a portable digital identity, replacing the dependence on a singular platform account. In practice, this authentication method is usually built using two technologies: a decentralised identifier controlled by the user and the verifiable credentials, which is a cryptographically signed claim such as proof that the user is the account holder. One of the main advantages of this method is the degree of control the user gets in regard to which credentials are presented. Furthermore, its portability means that the same credential can be used across multiple platforms. It also supports selective disclosure, meaning that a user can choose to reveal just the necessary information rather than their entire identity profile (Mazzocca et al., 2024). Compared with Unity Authentication, decentralized identity emphasizes user-controlled credentials and portability across platforms. However, Unity Authentication currently represents a centralized service model in which Unity and the developer share operational responsibilities. This difference is important for the present paper because it frames Unity Authentication as a practical commercial identity layer rather than a decentralized identity architecture.

Privacy-preserving authentication is another authentication system that checks the legitimacy of a user while exposing little personal data. This method works by reducing unnecessary disclosure during login. This means that instead of sending full profiles or raw biometric data, the system provides only what's needed for said session. In practice, this means that the system will involve selective disclosure of personal data, cryptographic proofs, or privacy-preserving biometric data (Sorrentino & López-Guzmán, 2025). A primary strength of this method is that it offers better privacy for users, does not contribute to data over-collection from third parties, and reduces exposure of sensitive biometric data (Sorrentino & López-Guzmán, 2025). It is however, a technically complex system to implement and deploy, significantly more complex than traditional authentication methods. Its focus on privacy and user anonymity can make it harder to support traceability and abuse investigations (Sorrentino & López-Guzmán, 2025). This literature is relevant to the present paper because it highlights the principle of data minimization. In Unity Authentication, this principle translates into avoiding unnecessary storage of personal data in developer-defined fields, limiting account-linking data, and treating persistent identifiers such as Unity Authentication Service IDs and external OIDC subject claims as personal data.

The novelty of this paper is not the proposal of a new authentication protocol. Instead, the contribution is a Unity-specific security and privacy assessment of existing authentication mechanisms used in immersive applications. Unlike broad metaverse-security surveys, this study focuses on the practical integration of Unity Player Accounts, OIDC-based sign-in, and Custom ID sign-in. Unlike general OAuth/OIDC security guidance, it examines how these authentication models appear in a Unity XR development context, including developer responsibilities, service-account risks, session behavior, account-linking risks, GDPR-relevant identifiers, and observable network endpoints during sign-in. The paper therefore contributes a practice-oriented analytical framework that connects authentication theory, Unity implementation details, and privacy-engineering interpretation.

Table 1. *Related Work Comparison and Positioning of the Present Study*

Study/ source	Scope	Platform or context	Authentication model	Privacy angle	Technical depth	Difference from the present paper
Hallal, Rhinelander & Venkat, 2024	Survey of authentication methods in XR	Extended reality environments	MFA, biometrics, general XR authentication	Discusses authentication risks and user-related security	Broad survey-level analysis	Broad XR authentication survey; does not analyze Unity Authentication implementation or traffic observations
Kang, Park & Kim, 2025	Continuous/ adaptive authentication for metaverse workspaces	Metaverse/ digital workspace	Behavioral biometric or adaptive MFA	Addresses continuous identity verification and contextual risk	Conceptual/ technical authentication model	Focuses on adaptive/ behavioral authentication rather than Unity Player Accounts, OIDC, or Custom ID
Kim et al., 2023	Privacy-preserving decentralized identity	Metaverse environment	Decentralized identifiers and verifiable credentials	Strong privacy and user-control focus	Cryptographic/ protocol-oriented	Proposes decentralized identity; this paper analyzes Unity's centralized commercial authentication service

Mazzocca et al., 2024	Survey of DIDs and verifiable credentials	General decentralized identity ecosystem	DID/VC-based identity	Selective disclosure and portability	Broad technical survey	Surveys DID/VC systems; does not address Unity or XR implementation
Sorrentino & López-Guzmán, 2025	Privacy risks for avatars and biometric/ inferred data	Metaverse privacy	Privacy-preserving authentication and data minimization	Strong focus on biometric and inferred personal data	Legal/privacy-oriented	Focuses on avatar and biometric privacy; this paper focuses on Unity authentication identifiers and developer responsibilities
OWASP Mobile guidance	Mobile/ client-side application security	Mobile and client-side apps	Secure storage, logging, session handling	Prevents leakage of sensitive data	Practical security testing guidance	Used as an evaluation lens, not as a Unity-specific study
RFC 9700 / OAuth 2.0 security guidance	OAuth/OIDC security best practices	Web, mobile, and public clients	OAuth 2.0 and OIDC	Token leakage, redirect, and client security risks	Protocol/ security best-practice level	Provides general OAuth/OIDC guidance; this paper applies those risks to Unity integration
Unity Authentication documentation	Unity Gaming Services authentication	Unity applications and games	Player Accounts, OIDC, Custom ID, anonymous sign-in	Unity/ developer responsibility and authentication identifiers	Product documentation	Describes features; this paper provides independent comparison, privacy interpretation, and traffic observation
Present paper	Unity-specific security and privacy assessment	Unity XR / immersive application	Unity Player Accounts, OIDC, Custom ID	GDPR-relevant identifiers, developer responsibility, metadata exposure	Practice-based implementation + comparative analysis + traffic observation	Unity-specific implementation-oriented assessment

The comparison shows that existing studies provide important background on authentication in XR and metaverse environments, but they generally remain broad, protocol-oriented, privacy-theoretical, or focused on alternative identity models such as biometrics and decentralized identity. The present study is narrower but more implementation-specific: it examines how Unity Authentication mechanisms are configured and assessed in an immersive application, and how developer responsibilities emerge across Unity Player Accounts, OIDC, and Custom ID sign-in.

RESEARCH DESIGN AND METHODOLOGY

This study adopts a structured practice-based security assessment methodology. The objective is not to evaluate the cryptographic strength of Unity Authentication itself, but to assess the security and privacy implications of integrating Unity Player Accounts, OpenID Connect, and Custom ID sign-in in a Unity-based immersive application. The study combines documentation analysis, prototype implementation, and network traffic observation.

The analysis is guided by the following research questions:

RQ1: What security and privacy responsibilities remain with the developer when Unity Authentication is integrated into an immersive application?

RQ2: What network endpoints and authentication-related communication patterns are observable during Unity Player Accounts sign-in?

RQ3: What implementation risks arise from token handling, account linking, session persistence, and service account management?

RQ4: How do Unity Player Accounts, OIDC-based sign-in, and Custom ID sign-in compare with respect to trust boundaries, developer responsibility, privacy exposure, and backend security requirements?

Based on these questions, the study evaluates the authentication approaches using the following criteria: client-side token exposure risk, backend trust assumptions, identity-provider dependency, session persistence behavior, account-linking risk, service-account exposure, GDPR-relevant personal data processing, and implementation complexity.

The methodology consists of four stages. First, Unity Authentication documentation and relevant security/privacy guidance are reviewed to identify expected trust boundaries and responsibilities. Second, a Unity XR prototype is implemented using Unity Player Accounts authentication. Third, authentication-related traffic is observed with Wireshark during the Unity Player Accounts sign-in workflow. Fourth, the results are interpreted using a threat-oriented analytical framework that maps observed behavior and implementation characteristics to security and privacy risks.

ASSUMPTIONS AND EXPECTED FINDINGS

The study is based on the following working assumptions:

H1: Unity Player Accounts authentication communicates with Unity and external identity-provider infrastructure over HTTPS/TLS, limiting passive inspection of authentication payloads but still exposing metadata such as destination domains, timing, and connection patterns.

H2: The main practical risks in Unity Authentication integrations are not weaknesses in TLS, but implementation-level weaknesses such as token leakage, insecure local storage, excessive logging, unclear account-linking flows, and insufficient backend hardening.

H3: Custom ID sign-in reduces direct client-side identity assertion risk by requiring a server-authoritative backend, but it introduces additional operational risk through privileged service accounts and backend token issuance logic.

H4: From a privacy perspective, persistent authentication identifiers and external identity claims should be treated as personal data, especially when linked to user profiles, telemetry, saved progress, or third-party identity providers.

COMPARATIVE SECURITY AND PRIVACY ANALYSIS OF UNITY AUTHENTICATION APPROACHES

Privacy and Security overview

Modern games and interactive applications rely on identity services not only to authenticate players, but also to enable cross-device progression, personalization, and secure access to backend features. Unity Authentication, part of Unity Gaming Services (UGS), supports multiple sign-in methods and includes a “Bring your own identity” (Unity Technologies, Approaches to authentication, n.d.a) option in two forms:

OpenID Connect (OIDC) (Unity Technologies, OpenID Connect, n.d.b) integration and Custom ID sign-in (Unity Technologies, Custom ID sign-in, n.d.c). The purpose of this section is to describe the security and privacy implications of these two approaches, to clarify the division of responsibilities between Unity and the developer, and to outline practical measures that reduce common risks such as token leakage, account takeover, identity confusion during linking, and misuse of privileged service credentials.

This section also addresses privacy with GDPR considerations. Unity Authentication creates and processes identifiers (for example, the Unity Authentication Service ID), and can be connected to external identity systems that provide additional identifiers, such as the OIDC subject claim. Because authentication data can be personal data (Unity Technologies, Privacy overview, n.d.d), privacy requirements must be treated as first-class design constraints and not as an afterthought.

Threat model: why authentication in games is special

A security design for authentication in a Unity client must start from the assumption that the game client is a public client, meaning it runs on hardware controlled by the user and can be inspected, modified or instrumented (Internet Engineering Task Force, Best Current Practice for OAuth 2.0 Security (RFC 9700), 2023). Even in honest user scenarios, crash logs, analytics traces, or debugging output can unintentionally expose sensitive values. In adversarial scenarios, attackers may attempt to steal tokens from local storage, intercept traffic using local proxies, perform replay attacks, abuse session refresh mechanisms, or trick linking flows into binding the wrong external identity to an existing account. Guidance for mobile and client-side security repeatedly highlights the risk of insecure local storage, over-permissive logging, and weak session handling, all of which are directly relevant to token-based sign-in approaches (OWASP Foundation, OWASP Mobile Top 10, n.d.a).

At the same time, the server side becomes a high-value target because it often holds privileged credentials and makes authoritative security decisions. In Unity's Custom ID approach, the backend requests tokens on behalf of the user (Unity Technologies, Custom ID sign-in, n.d.c). As a result, the backend must be designed with rate limiting, input validation, secret management, and auditing, because a compromise here can undermine the entire identity layer. In particular, any credentials used for Unity service-to-service access (for example, service accounts) must be protected as high-impact secrets (Unity Technologies, Admin authentication (Service accounts), n.d.f).

Security considerations for OpenID Connect in Unity Authentication

Responsibility split and trust boundaries

Unity's OIDC support is best understood as a bridge between an external identity provider and Unity Authentication. Unity provides a mechanism to configure an OIDC provider in the project and a client API to sign in using an OIDC token, but it does not implement the full identity provider integration on the developer's behalf (Unity Technologies, OpenID Connect, n.d.b). In other words, obtaining a valid ID token is typically outside of Unity Authentication's scope. It is the developer's responsibility to implement the login experience with the chosen identity provider and then pass the resulting token into Unity for sign-in. This division of responsibilities is important because the overall safety of the flow depends on how the token is obtained, transported, validated, and stored (Internet Engineering Task Force, Best Current Practice for OAuth 2.0 Security (RFC 9700), 2023) before it reaches Unity's authentication pipeline.

The OIDC configuration includes an issuer URL, and Unity requires the issuer to use HTTPS (Unity Technologies, OpenID Connect, n.d.b). From a security standpoint, this requirement enforces encrypted transport and helps ensure that tokens are issued by a trusted authority rather than by an endpoint susceptible to

trivial interception. However, HTTPS alone is not sufficient. Developers still need to treat the identity provider as a trust boundary and ensure their application does not accept tokens from unintended issuers or redirect flows to untrusted domains.

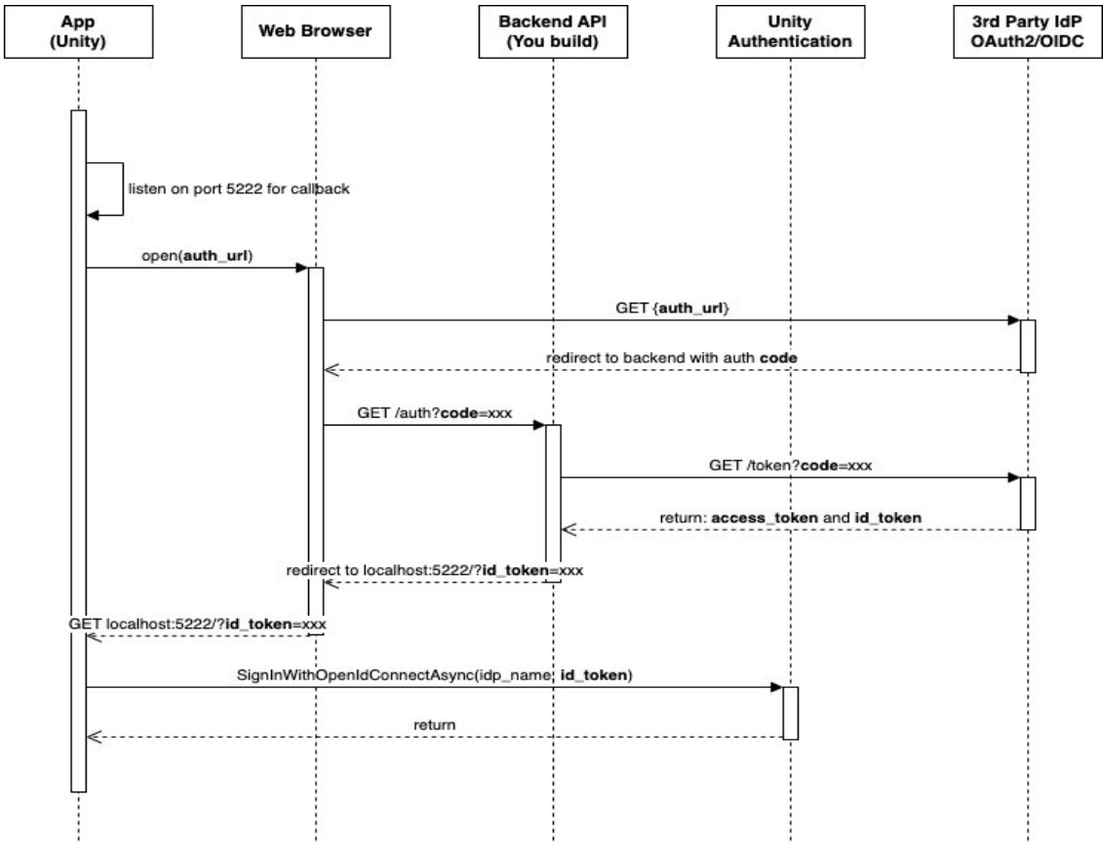

Figure 1. *UML sequence diagram with a typical integration between the third-party IdP, Unity Gaming Services, and the Unity app/game. Source: Unity Docs (Unity Technologies, OpenID Connect, n.d.b)*

Token exposure risks and secure handling

OIDC sign-in relies on tokens that represent identity and authorization. Once a token is exposed, an attacker may be able to impersonate the user until the token expires or is revoked, depending on the system's design. Therefore, token exposure is frequently a more practical attack vector than cryptographic breaking of the token itself (Internet Engineering Task Force, Best Current Practice for OAuth 2.0 Security (RFC 9700), 2023). In a Unity application, token exposure can occur through debug logs, analytics events, crash reports, client-side storage in plain text, or accidental sharing across components (OWASP Foundation, OWASP Mobile Top 10, n.d.a). Client security guidance emphasizes avoiding leakage of sensitive data through logs and ensuring that any sensitive data stored locally is protected against inspection by users, malware, or tooling (OWASP Foundation, Testing Data Storage (MASTG), n.d.b).

A secure Unity practice is to avoid printing tokens to the console, to avoid sending tokens to third-party telemetry systems, and to store any session-related values only when strictly necessary. When persistence is required, platform secure storage mechanisms should be used (for example, OS-provided key stores). Testing practices from mobile security literature

recommend explicitly validating that sensitive values are not written to local files, shared preferences, or logs, and that the app behaves safely under debugging and instrumentation (OWASP Foundation, Testing Data Storage (MASTG), n.d.b).

Account linking and identity confusion

A notable functional feature in Unity Authentication is account linking, where an anonymous account can later be upgraded and linked to an external identity (Unity Technologies, Anonymous authentication & linking, n.d.g). The security risk in such flows is identity confusion, where the wrong identity is linked due to user error, UI ambiguity, or malicious manipulation of tokens. From a design perspective, linking should be treated as a sensitive operation requiring explicit user intent, clear UI confirmation, and careful handling of any cached or background tokens. Unity's OIDC documentation includes explicit linking APIs (for example, linking an existing session with an OIDC identity), which underlines that this is a first-class capability rather than an incidental feature (Unity Technologies, OpenID Connect, n.d.b).

From a privacy perspective, linking can also expand the amount of personal data being processed, because the system may now associate the Unity Authentication identifier with an external identity claim such as the OIDC subject. Unity's privacy documentation explicitly references that an external identifier can be an OpenID Connect subject claim (Unity Technologies, Privacy overview, n.d.d), which is important for transparency and data mapping in privacy notices.

Security considerations for Custom ID sign-in (server-authoritative design)

Server-authoritative flow (why Unity insists on a backend)

Custom ID sign-in is intended for cases where developers already maintain a user identity system or need a project-specific identifier that does not map directly to a consumer identity provider. Unity's documentation states that the game client cannot request Unity Authentication tokens directly in this model and requires a backend that is "server-authoritative". The reasoning is straightforward: if a client could claim an arbitrary custom ID and obtain valid tokens without server validation, it would be easy to impersonate other users (Unity Technologies, Custom ID sign-in, n.d.c). Requiring a backend places the identity decision behind a controlled environment where secrets can be protected and where the developer can implement verification logic, risk checks, and abuse prevention.

Token lifecycle, session reuse, and logout

With Custom ID sign-in, the player still ends up with the same Unity Authentication session model: a short-lived token used as the UGS access token (about one hour) plus a longer-lived session token cached on the device; in Unity's responses, this short-lived token is returned as idToken, but it functions as the access token in this context. By default, signing out clears the access token but keeps the session token so the player can re-authenticate without a fresh login. If you need a true "fresh start" on the device (for example, shared devices or account switching), Unity provides options to clear the session token (for example, SignOut(true) or ClearSessionToken()). The trade-off is usability versus recoverability: clearing a session token can strand an anonymous account if it has not been linked to an external identity (Unity Technologies, Unity Authentication sessions, n.d.h).

Protecting service accounts and privileged credentials

The core security advantage of Custom ID sign-in server-side control is also its biggest risk if mismanaged. The backend uses a Unity service account to authenticate to Unity services and issue tokens for players. Unity's service

account documentation describes how service accounts authenticate (Unity Technologies, Admin authentication (Service accounts), n.d.f), and Unity's Custom ID guidance further notes that the service account must be granted the Player Authentication Token Issuer role in order to issue player authentication tokens (Unity Technologies, Custom ID sign-in, n.d.c). In practice, this means that if the service account secret is leaked, an attacker may gain the ability to issue tokens or access administrative endpoints, depending on the roles granted (Unity Technologies, Admin authentication (Service accounts), n.d.f). Least-privilege assignment, secret rotation, and strong operational controls are therefore mandatory, not optional (Internet Engineering Task Force, Best Current Practice for OAuth 2.0 Security (RFC 9700), 2023). Service account secrets must never be embedded in a client build, never be stored in source control, and should be managed using a secret manager with restricted access.

Backend endpoint hardening

Because the backend acts as the authoritative control point, it must be resilient to abuse. A practical threat in Custom ID scenarios is enumeration and brute forcing of IDs, especially if IDs are predictable. Backend endpoints should implement rate limiting (OWASP Foundation, OWASP Mobile Top 10, n.d.a), anomaly detection, and strict validation of ID formats. Where feasible, IDs should be high-entropy or bound to an authenticated session in the developer's identity system. Logging should be designed to support auditing without capturing sensitive token material (OWASP Foundation, Testing Data Storage (MASTG), n.d.b). These measures are consistent with broadly accepted application security principles for protecting authentication tokens and reducing the impact of token misuse, and they align with client security guidance on preventing sensitive data leakage through logs and insecure storage.

Privacy considerations: data types, legal roles, and user rights

What data is processed in Unity Authentication

Unity Authentication documentation indicates that a unique Unity Authentication Service ID (UAS ID) is created for users, and that additional identifiers may be processed depending on the sign-in method. For example, when using external identity providers, Unity can process external identifiers such as an OpenID Connect subject claim. The privacy implication is that even if the application does not directly collect a real-world name, authentication identifiers can still be personal data when they relate to an identifiable individual or can be used to single out a user across sessions (Unity Technologies, Privacy overview, n.d.d). Therefore, developers should model these identifiers as personal data within their internal records and privacy documentation.

Unity further advises that developers should not store personal data in "developer defined data" fields because Unity's systems will not treat such values as personal data for retention and data subject request handling. This point is particularly relevant for games, where developers may be tempted to store emails, phone numbers, or other identifiers for convenience. From a privacy-engineering perspective, such storage practices create compliance risk because deletion or export requests may not capture the data correctly (European Data Protection Board, Guidelines 07/2020 on the concepts of controller and processor in the GDPR, 2020).

Controller/processor roles and accountability under GDPR

Unity's privacy overview for Authentication states that, under GDPR, Unity acts as a processor while the developer acts as the controller. This means the developer determines the purposes and means of processing, including what sign-in methods are used, what identifiers are collected, how long data is retained, and what

downstream services are connected. Unity, as the processor, provides the service and processes data on the developer's behalf under contractual terms. The European Data Protection Board's guidelines on controller and processor concepts provide a broader legal framing for these roles and clarify that accountability for lawful basis, transparency, and rights handling rests primarily with the controller (European Data Protection Board, Guidelines 07/2020 on the concepts of controller and processor in the GDPR, 2020).

In practical terms, a Unity game that uses Unity Authentication should maintain a privacy notice that explains what identifiers are used, why they are processed, what third parties are involved (including Unity as a service provider), and how users can exercise rights such as access and deletion (Unity Technologies, Privacy overview, n.d.d). The notice should also reflect linking behavior, because linking can change the nature of the data (for example, associating an anonymous profile with an external identity) (Unity Technologies, Anonymous authentication & linking, n.d.g).

Data subject requests, deletion, and retention

Unity's privacy documentation indicates that Unity Authentication supports mechanisms for access and deletion requests, including developer-managed options and platform tooling (Unity Technologies, Privacy overview, n.d.d). From a privacy governance standpoint, it's important that these requests are handled end-to-end: deleting an authentication profile should also trigger deletion or anonymization across connected systems where feasible (for example, saves, inventories, or profiles stored in other services). Retention should be specified in developer policies and implemented consistently, rather than relying on indefinite storage by default (European Data Protection Board, Guidelines 07/2020 on the concepts of controller and processor in the GDPR, 2020).

Children's privacy and transparency requirements

Unity documentation highlights that certain UGS privacy constraints apply regarding child users and parental consent in jurisdictions where it is required. Even when an application does not target children, developers should consider age-related compliance in their distribution markets and ensure that authentication flows do not inadvertently encourage collection of data that triggers additional obligations (European Data Protection Board, Guidelines 07/2020 on the concepts of controller and processor in the GDPR, 2020). Transparency remains essential: Unity emphasizes that developers should maintain their own privacy policy for the application rather than relying on Unity's policy as a substitute, because the developer is responsible for describing the complete data processing context of the application (Unity Technologies, Privacy overview, n.d.d).

Table 2. *Comparative Security and Privacy Analysis of Unity Authentication Approaches*

Criterion	Unity Player Accounts	OIDC via external IdP	Custom ID sign-in
Primary trust anchor	Unity account infrastructure	External identity provider + Unity Authentication	Developer backend + Unity Authentication
Token acquisition responsibility	Unity-managed login flow	Developer/IdP integration before Unity sign-in	Developer backend issues/signs request
Backend required	No, for basic use	Usually optional, depending on architecture	Yes
Client-side token exposure risk	Medium	Medium/High if tokens handled manually	Lower for identity assertion, still present for session tokens
Service-account risk	Low for basic use	Depends on backend use	High if service credentials are mishandled
Account-linking risk	Present	Present, especially with external identity claims	Present if custom IDs map incorrectly
Privacy exposure	Unity identifier, linked account data	Unity identifier + external subject claim	Unity identifier + developer-defined identifier
Best fit	Fast integration, consumer login	Existing IdP or enterprise identity	Existing backend identity system
Main security concern	Session handling and account linking	Token acquisition/ storage and redirect trust	Backend hardening and service-account protection

Unity Authentication's OIDC and Custom ID approaches enable flexible identity architectures, but they shift critical security and privacy responsibilities onto the developer (Unity Technologies, Approaches to authentication, n.d.a). OIDC integration is secure when token acquisition and handling follow modern OAuth/ OIDC best practices and when tokens are treated as sensitive secrets in a hostile client environment (OWASP Foundation, Testing Data Storage (MASTG), n.d.b). Custom ID sign-in is safer against impersonation because Unity enforces a server-authoritative model, yet it raises the operational security bar by introducing privileged service accounts and a backend that must be secured against abuse (Unity Technologies, Admin authentication (Service accounts), n.d.f). From a privacy perspective, authentication identifiers and external claims should be treated as personal data, documented transparently, and managed with clear retention and deletion procedures. Ultimately, the safest implementations are those that combine strict secret handling, hardened server endpoints, minimal data collection, and well-defined user rights workflows aligned with Unity's guidance

and GDPR-oriented controller obligations (Unity Technologies, Privacy overview, n.d.d).

Prototype Implementation / Case Study: Unity Player Accounts in XR

Authentication is a critical component for the metaverse, as it manages user data in a way that allows for advanced security. Like in any other application, Unity offers 2 approaches for authentication: authenticating players anonymously and through external identity providers (Unity Technologies, Approaches to authentication, n.d.a). In addition, it provides a best practices guide, which allows us to decide more easily which authentication method to integrate depending on the characteristics of the metaverse we are building.

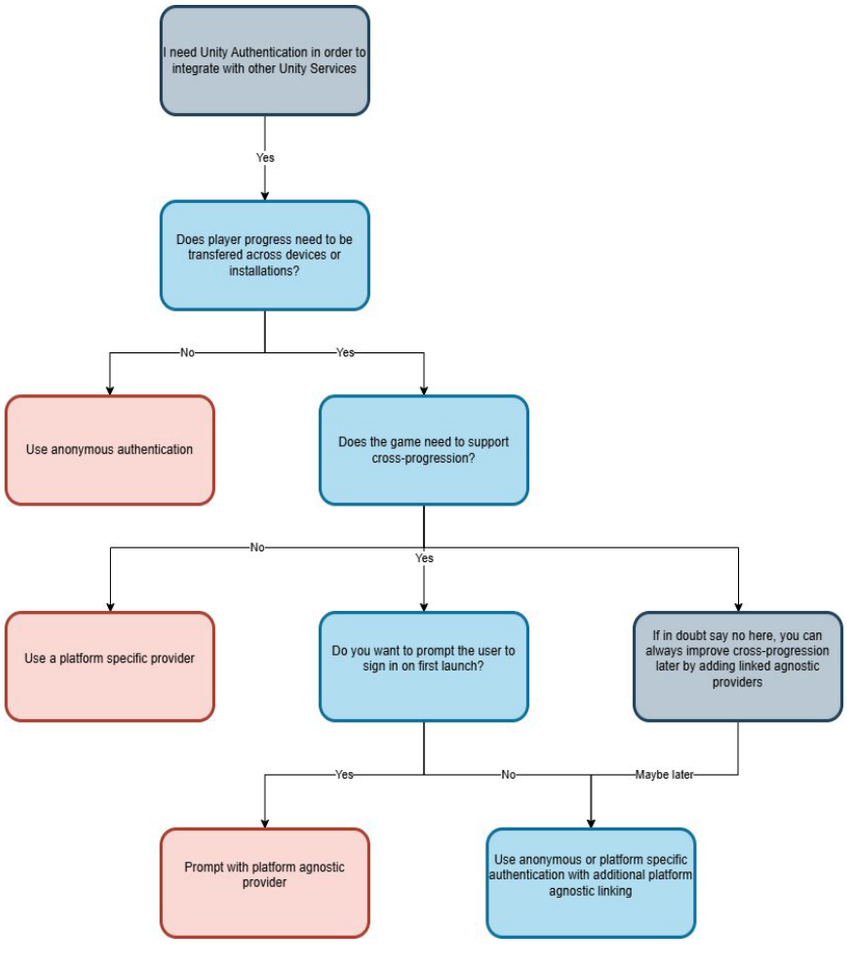

Figure 2. *Configuration Workflow for Unity Player Accounts Authentication in the XR Prototype*

Steps required to configure authentication in the metaverse through Unity using the Unity Player Accounts provider.

- The project must be connected to the cloud provided by Unity. We can accomplish this step when we create the project in Unity Hub by checking: **Connect to Unity Cloud** or directly from Unity: **Edit → Project Settings → Services.** In the first case, a new project will be generated in the cloud. In the second case, we have two options: connecting to an already existing project in the cloud or creating a new project in the cloud to which we assign the created project.

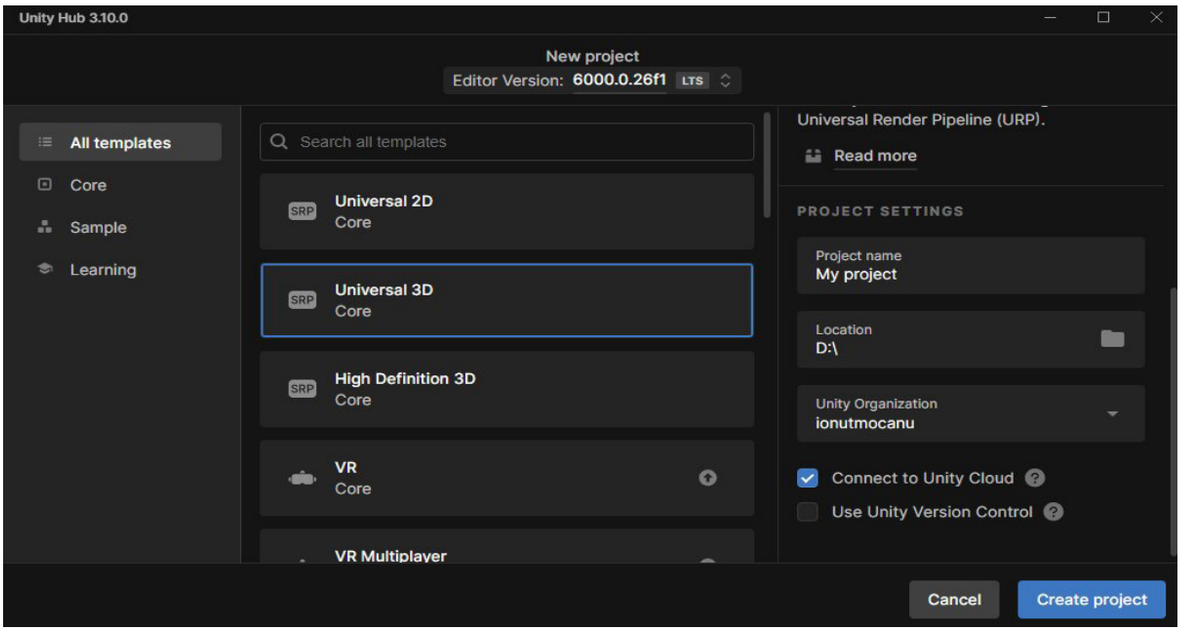

Figure 3. *Connecting to the Cloud Directly in Unity Hub*

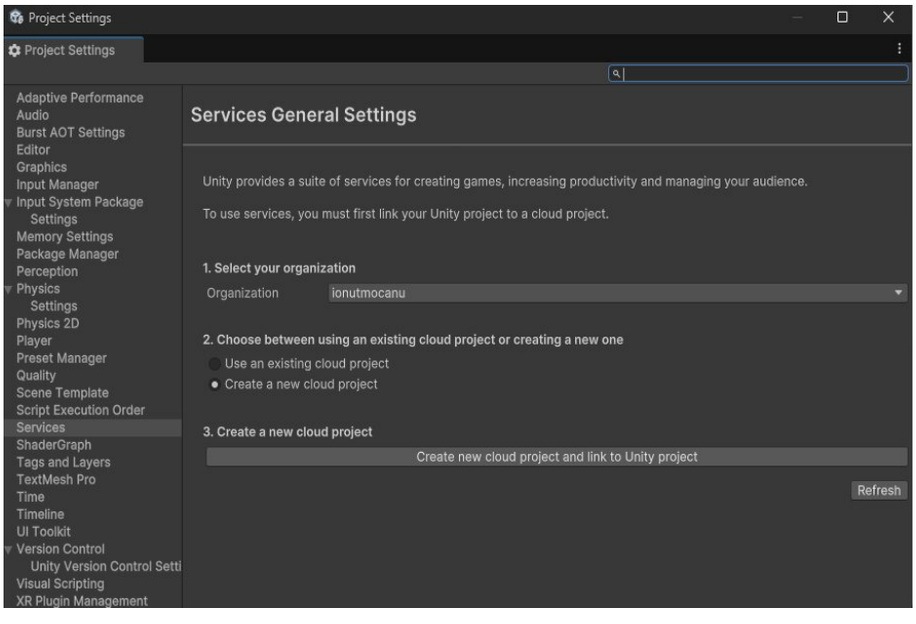

Figure 4. *Connecting to the Cloud Directly from Unity. Own Source*

- Setting the ID provider in Unity from **Unity Dashboard → the desired project → Player Authentication → Add Identity Provider → Unity Player Accounts**. In the new panel opened on the screen, we check both platforms: **iOS / Android, PC.**

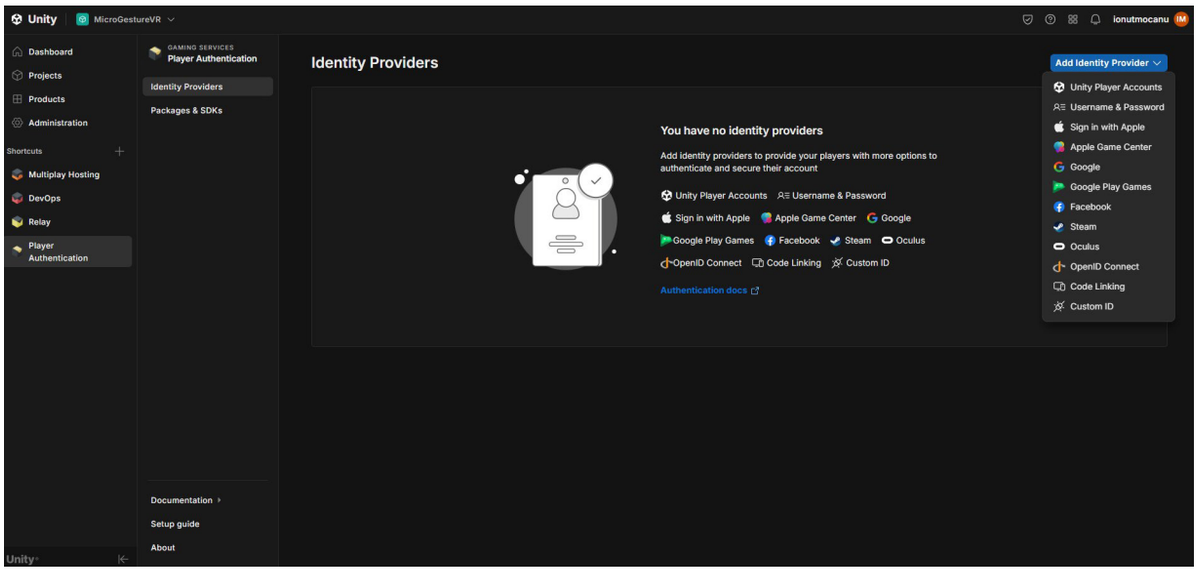

Figure 5. *Player Authentication in Unity Dashboard*

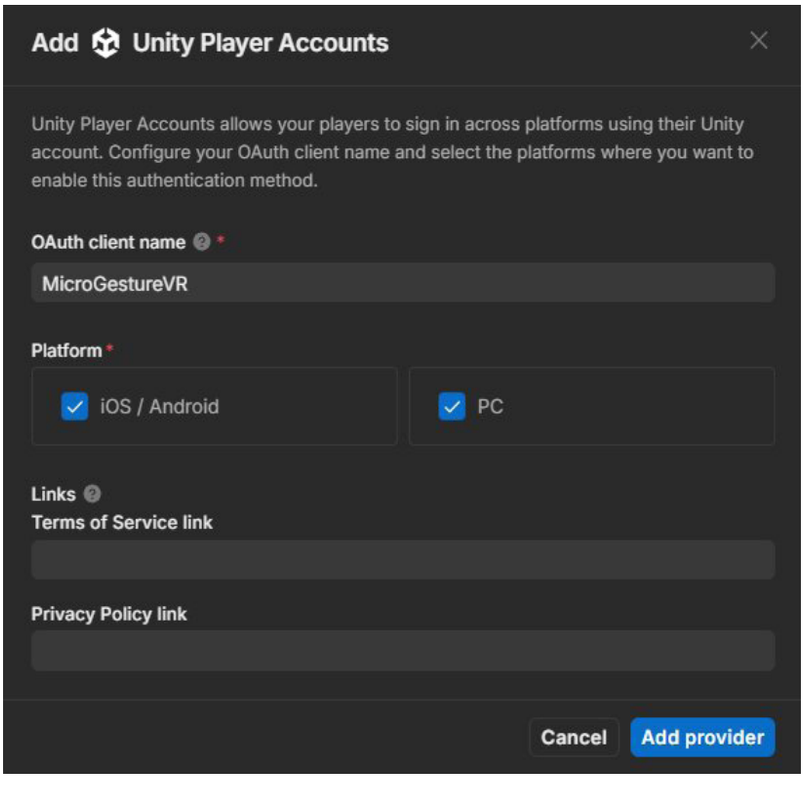

Figure 6. *Unity Player Accounts Provider Configuration in the Unity Dashboard*

- Install the **Authentication** package from **Package Manager → Services.**
- Change the project profile in Unity. For this step, go to **Unity to File → Build Profiles → Android** and select **Switch Platforms.**
- In the same menu, under **Platform Settings → Run Device,** select the headset that appears. If it does not appear, select **Refresh.**

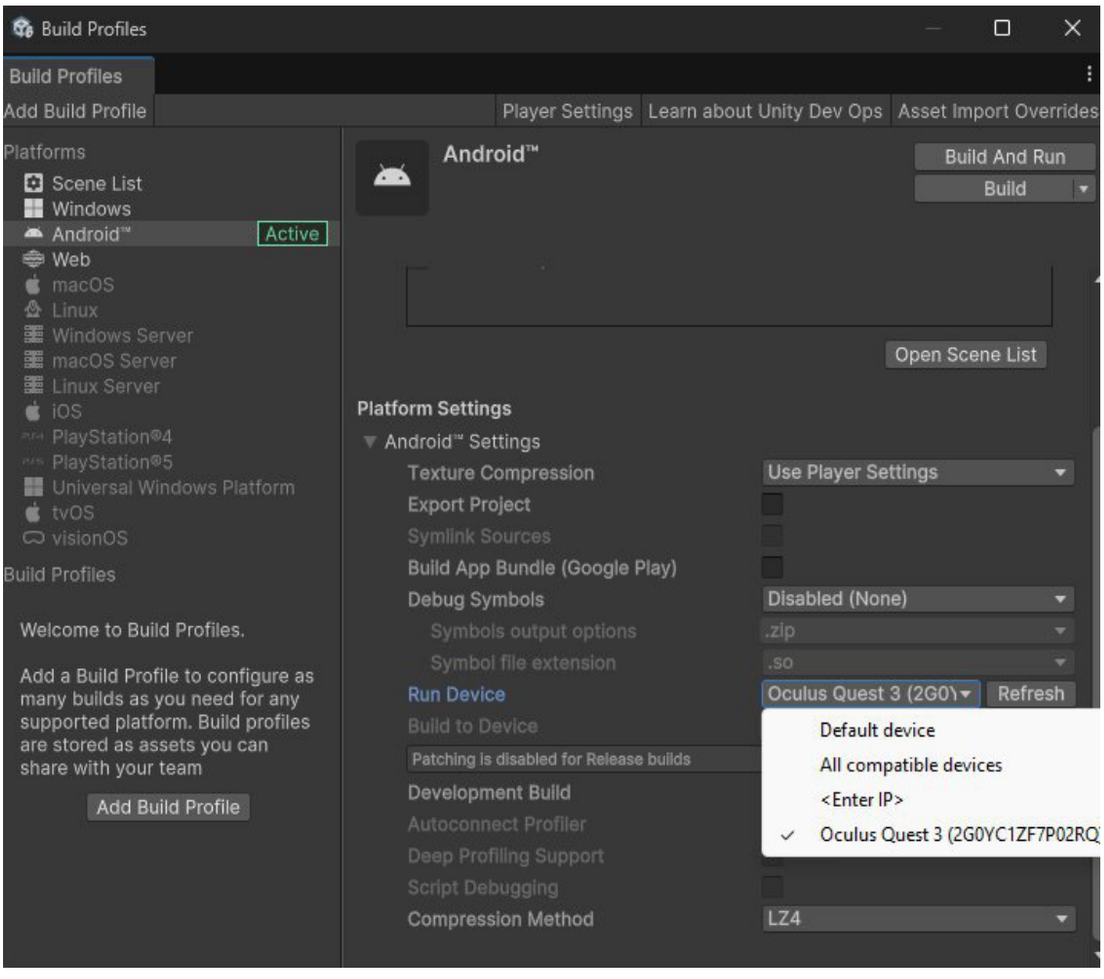

Figure 7. *Selecting the Headset in Build Profiles*

- Check whether the client ID provided by Unity is entered in the **Client ID** field in **Unity Player Accounts → Services → Project Settings**.
- Testing. For this step, we have the code provided in (Unity Technologies, *Approaches to authentication*, n.d.a) available to quickly test whether the entire setup works properly.

Implementation of authentication in the metaverse based on Unity Player Accounts

In this article, we developed a small platform that allows login through Unity. In order to provide an immersive experience appropriate for the metaverse, we also focused on the user experience. Thus, in the figure below, the authentication menu can be observed with a clean design and clear instructions for any age group. For this platform, we started from Meta's SDK for Unity: **Meta XR All-in-One** (Unity Technologies, *Meta XR All-in-One SDK*, n.d.i). It is free and can be accessed from the Unity Asset Store, and the official documentation can be found on Meta's website.

The workflow for connecting the user to the Unity account from the metaverse is as follows: click the Sign In button. Another panel will appear that provides the user with instructions to remove the headset and connect through the default browser tab that has just opened. After the login process is completed, we can observe the player's ID in Player Management from the Unity Dashboard.

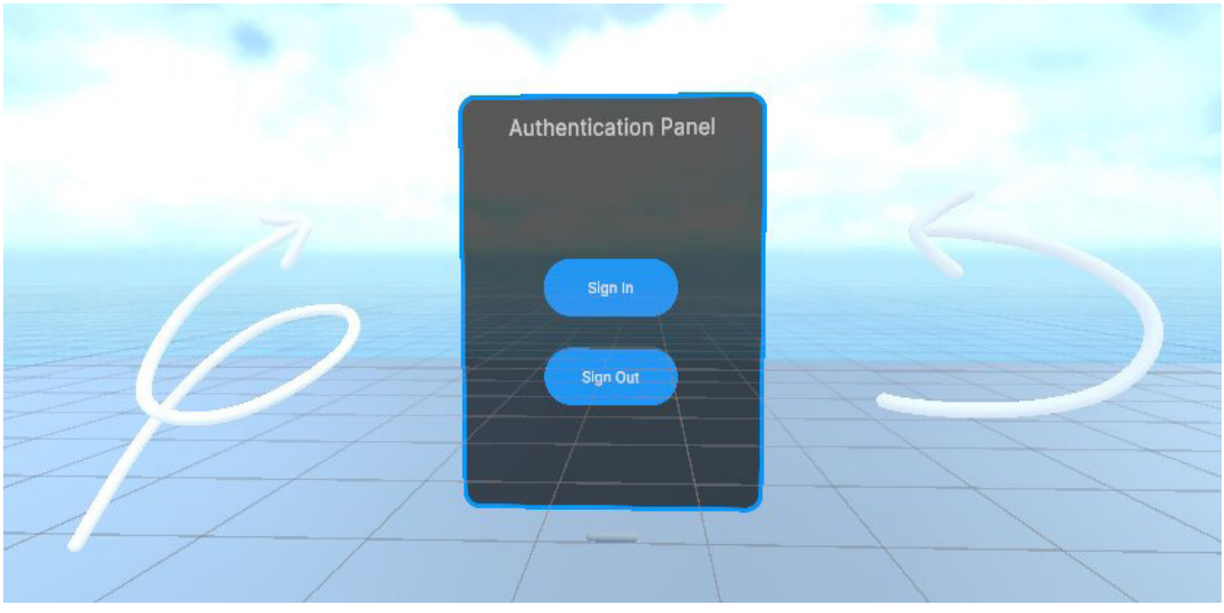

Figure 8. *Connecting from the Metaverse*

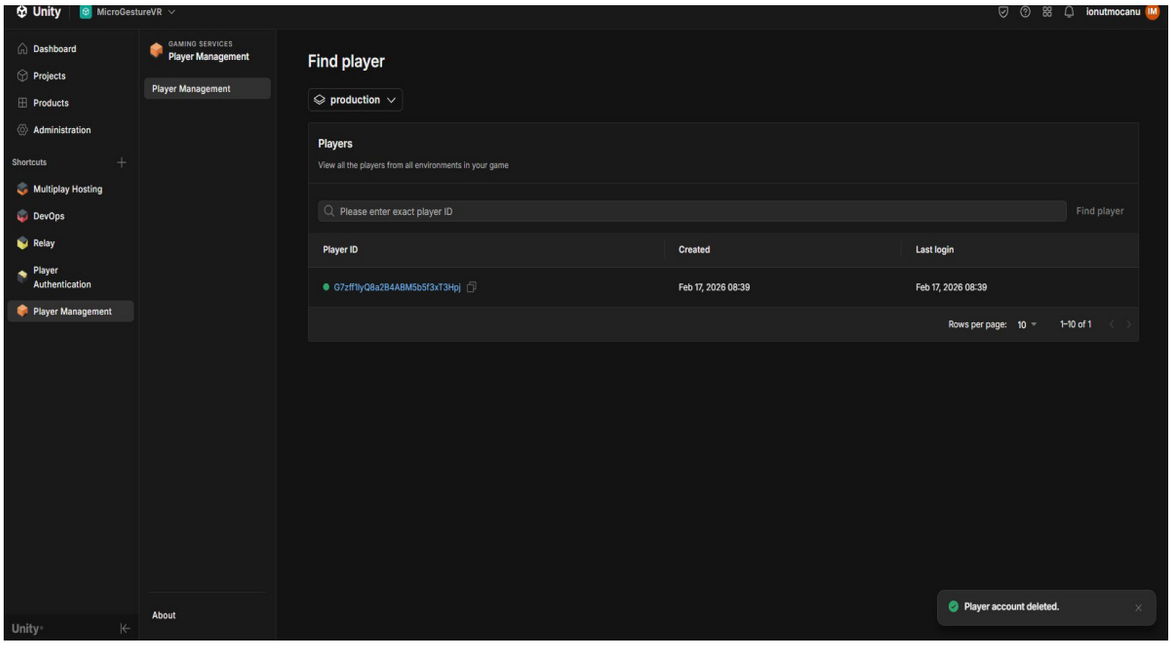

Figure 9. *Observing the Newly Created ID*

Traffic Analysis

For this project, we chose to use Wireshark, which is a free network protocol analyzer. Wireshark relies on a capture library installed on the computer in order to collect data. We chose this program because it is stable and has the capability to analyze hundreds of protocols (Computer Networking: A Top-Down Approach, 2017).

At the core of Wireshark, there is a packet sniffer. This is a tool that captures messages sent or received on a computer. It is important to mention that the packet sniffer is a passive

tool, which does not transmit data, and in addition, the packets are not directly addressed to it; it only receives a copy of them. It consists of two main elements: a packet capture library and a packet analyzer (Computer Networking: A Top-Down Approach, 2017).

In order to understand the data resulting from traffic analysis, it is important to know which protocols are used by the chosen authentication method. Traffic to Unity Player Accounts was observed as HTTPS traffic protected by TLS. TLS is not an improved version of TCP; rather, it is a security protocol layered above a transport protocol such as TCP in the conventional HTTPS stack. TLS provides confidentiality, integrity, and endpoint authentication for application-layer communication. As a result, passive packet capture can reveal metadata such as IP addresses, domain names through DNS or TLS Server Name Indication when visible, certificate information, timing, packet sizes, and connection sequences, but it cannot directly reveal encrypted application payloads such as credentials or tokens without additional privileged instrumentation or decryption keys.

Therefore, the goal of the traffic analysis in this study is not to inspect user credentials or token contents, but to identify authentication-related endpoints, communication phases, and observable metadata that are relevant for security and privacy interpretation.

In this case, the communication session takes place between the user represented by the player and the server represented by the Unity ID provider. The server holds a private/public key and a certificate that assigns its identity to the public key.

Based on the packet capture performed in Wireshark, we identified the essential servers that allow the Sign In process to take place. Below we detail each server: what it does and the code segment that triggers it:

- api.unity.com: The main gateway for the entire Unity cloud ecosystem. It is triggered in the Awake function through await UnityServices.InitializeAsync();, which performs a ping to initialize the project based on the project ID.
- player-login.unity.com: The web server that hosts the page from which the user logs in. It is triggered by PlayerAccountService. Instance.StartSignInAsync();, which opens the browser at the respective URL and passes the application parameters.
- player-auth.services.api.unity.com: acts as an authorization server according to the OAuth 2.0 standards. It is the one that makes the final decision: whether the login was successful and issues the access token (JWT). It is called by the browser after the credentials are entered.
- accounts.google.com: Google's authentication server, since in the login window that appeared based on step 2 we chose to sign in with Google through Unity.
- packages.unity.com: The editor checks whether there are new versions for the packages used by Unity.
- plausible.it.unity3d.com: analytics service used to count visits.

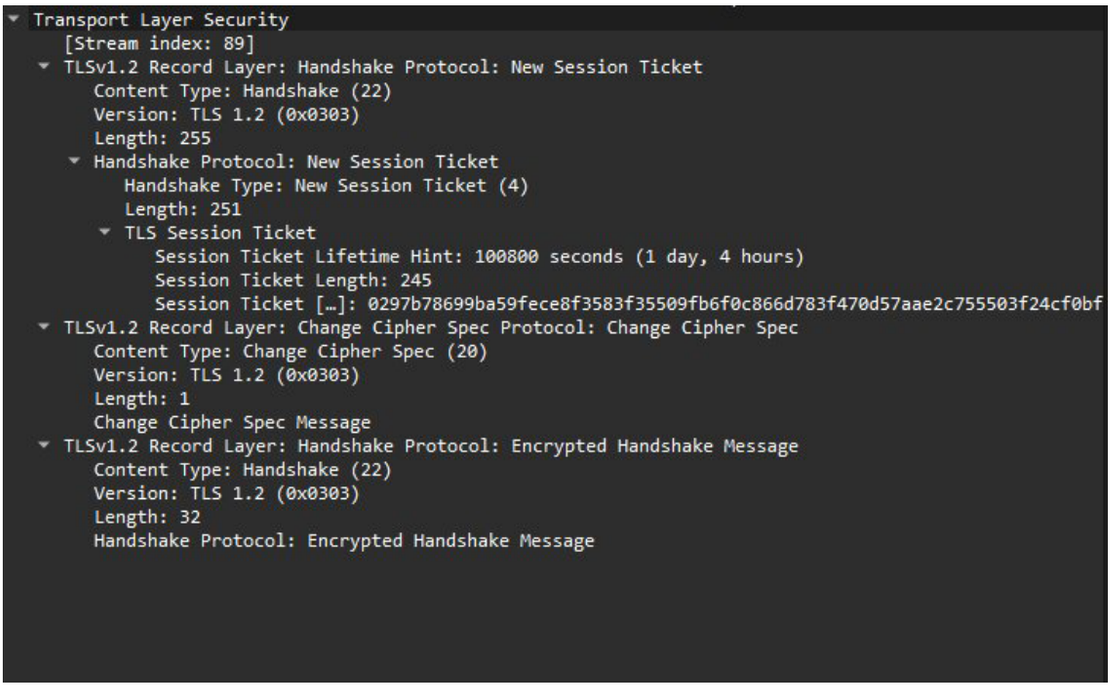

Figure 10. *Content of the TLS Section in a "New Session Ticket" Message Between Server and Client*

Table 3. *Traffic-Analysis Results and Security/Privacy Interpretation*

Observation	Result	Security interpretation	Privacy interpretation
Transport protection	Authentication traffic was observed as HTTPS/TLS-protected	Credentials and token payloads were not visible in passive capture	Payload confidentiality is preserved against passive local observation
Unity infrastructure visibility	Unity-related domains such as api.unity.com, player-login.unity.com, and player-auth.services.api.unity.com were observed	The authentication flow depends on Unity cloud infrastructure	Metadata reveals the use of Unity Authentication services
External IdP visibility	accounts.google.com was observed when Google sign-in was selected	The flow depends on the selected external identity provider	Metadata may reveal the identity-provider ecosystem used by the application
Payload visibility	No plaintext credentials or token payloads were observed	Passive traffic capture alone cannot inspect token contents	Privacy-sensitive content is protected, but metadata remains observable

Non-authentication Unity traffic	Additional Unity endpoints such as package or analytics-related services may appear	These should be separated from the core authentication path	Non-authentication telemetry or service traffic may create additional privacy considerations
Account-linking risk	Present	Present, especially with external identity claims	Present if custom IDs map incorrectly
Analytical limitation	TLS prevents direct packet-level inspection of application payloads	Secure token handling must be assessed through logs, storage inspection, and code review	Traffic analysis cannot determine whether local storage or application logs expose personal data

Discussion

The analysis shows that Unity Authentication security depends less on the confidentiality of network transport and more on implementation decisions around token handling, session persistence, account linking, and backend responsibility. Passive traffic capture confirmed the use of HTTPS/TLS and allowed authentication-related infrastructure to be identified, but it did not expose credentials or token payloads. This means that Wireshark-based analysis is useful for mapping communication dependencies and metadata exposure, but it cannot prove that tokens are stored securely or that account-linking logic is safe.

The comparison of Unity Player Accounts, OIDC-based sign-in, and Custom ID sign-in also shows that the approaches differ mainly in trust boundaries and operational burden. Unity Player Accounts offers a direct integration path for XR applications, while OIDC is more suitable when an external identity provider must be integrated. Custom ID sign-in gives the developer stronger control through a server-authoritative model, but this also increases responsibility for backend hardening, service-account protection, auditing, and abuse prevention.

From a privacy perspective, the most important conclusion is that authentication identifiers should be treated as personal data when they can identify, single out, or link users across sessions or services. This applies not only to external OIDC claims, but also to Unity Authentication identifiers and developer-defined identifiers used in Custom ID flows.

Limitations

This study has several limitations. First, the implementation is a single XR proof of concept using Unity Player Accounts and does not represent all possible Unity Authentication configurations. Second, the traffic analysis is limited by HTTPS/TLS encryption: passive packet capture can identify endpoints, timing, certificates, and connection metadata, but cannot inspect encrypted credentials, authorization codes, or token payloads. Third, the study does not perform cryptographic testing of Unity Authentication or benchmark security outcomes quantitatively. Fourth, the comparison of OIDC and Custom ID sign-in is based on documentation analysis and threat modeling rather than full parallel implementations of both approaches. Finally, the observations may vary depending on Unity package versions, headset firmware, operating system, identity-provider configuration, and network environment.

CONCLUSIONS

This study assessed Unity Authentication approaches for immersive applications using a structured practice-based methodology. The analysis showed that Unity Player Accounts provides a relatively direct integration path for XR applications, while OIDC and Custom ID sign-in introduce different trust boundaries and developer responsibilities. Passive Wireshark observation confirmed the use of HTTPS/TLS-protected communication and allowed the identification of authentication-related infrastructure, but it did not expose credentials or token payloads. This demonstrates both the value and the limitation of traffic analysis: it can validate communication patterns and metadata exposure, but it cannot by itself prove secure token handling inside the client.

The main security and privacy risks identified in this study are therefore implementation-level risks: token leakage through logs or local storage, unclear account-linking flows, session persistence on shared devices, excessive collection of authentication identifiers, and improper handling of privileged service-account credentials. Custom ID sign-in offers stronger server-authoritative control, but only when the backend is properly hardened and service credentials are protected. From a GDPR perspective, Unity identifiers, external identity-provider claims, and persistent session-related identifiers should be treated as personal data when they can identify or single out a user.

Future work should extend this study through larger-scale repeated captures, controlled comparison across multiple identity providers, automated local-storage inspection, and systematic testing of failed sign-in, account switching, token expiration, and account-linking scenarios.

ACKNOWLEDGEMENTS

This study is supported by the project “Advanced research based on emerging and disruptive technologies—support for the society of the future—FUTURE TECH” (funded by the Romanian Core Program within the National Research Development and Innovation Plan 2022–2027 of the Ministry of Research and Innovation), project no 23380601.

REFERENCE LIST

Computer Networking: A Top-Down Approach. (2017) Computer Networking: A Top-Down Approach. Global Edition. *Pearson.*

European Data Protection Board. (2020) *Guidelines 07/2020 on the concepts of controller and processor in the GDPR.* Available at: https://www.edpb.europa.eu/our-work-tools/our-documents/guidelines/guidelines-072020-concepts-controller-and-processor-gdpr_en.

Hallal, L., Rhinelander, J. and Ramesh, V. (2024) Recent trends of authentication methods in extended reality: A survey. *Applied System Innovation*, 7(3), 45. Available at: https://www.mdpi.com/2571-5577/7/3/45.

Internet Engineering Task Force. (2023) *Best Current Practice for OAuth 2.0 Security (RFC 9700).* Available at: https://datatracker.ietf.org/doc/rfc9700/.

Kang, G., Park, J. and Kim, Y.-G. (2025) Continuous behavioral biometric authentication for secure metaverse workspaces in digital environments. *Systems*, 13(7), 588. Available at: https://www.mdpi.com/2079-8954/13/7/588.

Kim, M., Oh, J., Son, S., Park, Y., Kim, J. & Park, Y. (2023) Secure and privacy-preserving authentication scheme using decentralized identifier in metaverse environment. *Electronics*, 12(19), p. 4073. Available at: https://www.mdpi.com/2079-9292/12/19/4073.

Mazzocca, C., Acar, A., Uluagac, S., Montanari, R., Bellavista, P. & Conti, M. (2024) A Survey on Decentralized Identifiers and Verifiable Credentials. *arXiv* preprint, arXiv:2402.02455v1. Available at: https://arxiv.org/html/2402.02455v1.

OWASP Foundation (n.d.) *OWASP Mobile Top 10*. Available at: https://owasp.org/www-project-mobile-top-10/.

OWASP Foundation (n.d.) Testing Data Storage (MASTG). *OWASP Mobile Application Security Testing Guid*e. Available at: https://mas.owasp.org/MASTG/0x05d-Testing-Data-Storage/.

Sorrentino, G. & López-Guzmán, J. (2025) *Rethinking privacy for avatars: biometric and inferred data in the metaverse. Published 12 May 2025.* Available at: https://rua.ua.es/server/api/core/bitstreams/f040b6c8-855d-4731-b3ab-a3f6687ece87/content.

Unity Technologies. (n.d.) *Admin authentication (Service accounts). Unity Services Web API Docs.* Available at: https://services.docs.unity.com/docs/service-account-auth/.

Unity Technologies. (n.d.) *Anonymous authentication & linking. Unity Docs – Authentication.* Available at: https://docs.unity.com/en-us/authentication/anonymous-auth-and-linking.

Unity Technologies. (n.d.) *Approaches to authentication. Unity Docs – Authentication.* Available at: https://docs.unity.com/en-us/authentication/approaches-to-authentication.

Unity Technologies. (n.d.) *Client authentication. Unity Services Web API Docs.* Available at: https://services.docs.unity.com/docs/client-auth/.

Unity Technologies. (n.d.) *Custom ID sign-in. Unity Docs – Authentication*. Available at: https://docs.unity.com/en-us/authentication/platform-signin/custom-id.

Unity Technologies. (n.d.) *Meta XR All-in-One SDK. Unity Asset Store.* Available at: https://assetstore.unity.com/packages/tools/integration/meta-xr-all-in-one-sdk-269657.

Unity Technologies (n.d.) *OpenID Connect. Unity Docs – Authentication.* Available at: https://docs.unity.com/en-us/authentication/openid-connect.

Unity Technologies. (n.d.) *Privacy overview. Unity Docs – Authentication (Privacy & Consent).* Available at: https://docs.unity.com/en-us/authentication/privacy-and-consent/overview.

Unity Technologies. (n.d.) *Unity Authentication sessions. Unity Docs – Authentication.* Available at: https://docs.unity.com/authentication/session-management