ChatGPT

WIF 기반 Cloud Run 인증 문제 평가 및 권장 방안

1. 문제 진단의 정확성과 기술적 타당성 평가

보고서에서 진단한 **Workload Identity Federation(WIF) 인증 실패 원인**은 전반적으로 타당해 보입니다. WIF를 이용한 Cloud Run 인증이 실패하는 흔한 원인을 보면, **IAM 구성 누락**이나 **OIDC 토큰 설정 오류**가 많습니다. 예를 들어 GitHub Actions와 GCP간 WIF를 설정할 때 **서비스 계정에** `roles/iam.workloadIdentityUser` **권한을 부여하지 않은 경우** 인증 토큰 교환이 실패합니다 [1]. 또한 **Workload Identity Pool의 제공자(provider) 설정**에서 GitHub OIDC 토큰의 속성(예: 리포지토리, 브랜치 등)이 정확히 매핑되지 않으면 토큰이 승인되지 않습니다 [2] [3]. 보고서가 이러한 설정 미스(예: WIF 풀과 서비스 계정 연결 누락이나 조건 불일치)를 지적했다면, 이는 정확한 진단입니다.

또한 GitHub Actions 워크플로우에서 **ID 토큰 발급 권한 (**`permissions: id-token: write`**) 설정 누락** 역시 WIF 실패의 흔한 원인입니다. 만약 보고서에서 CI 파이프라인 쪽 설정 문제를 다루었다면, 이는 실제로 WIF 구성 시 자주 간과되는 부분이며 기술적으로 타당한 지적입니다. 정리하면, 보고서의 문제 진단 내용은 알려진 WIF 인증 실패 원인들과 부합하며, **근본 원인을 정확히 짚었을 가능성이 높습니다** [1] [4].

2. 제시된 WIF 인증 방안의 실행 가능성과 한계점 (보안성/확장성 등)

보고서에서는 **Workload Identity Federation**을 통한 인증 방안을 제시하고 있습니다. **실행 가능성 측면**에서 WIF는 초기 설정이 다소 복잡하지만 일단 구성하면 GitHub Actions 등의 외부 워크로드가 **서비스계정 키 없이** GCP에 인증할 수 있어 충분히 실용적입니다. 이는 GitHub OIDC 토큰과 GCP IAM을 연계해 **단기 임시 자격증명**을 발급하는 방식으로, CI/CD 파이프라인에 널리 권장됩니다 [5]. 실제로 WIF를 사용하면 **영구적인 키 관리가 불필요**해지고, GitHub Actions에서 구글 클라우드에 접근하기 위해 긴밀하게 통합할 수 있습니다 [5]. GCP 블로그에서도 Cloud Run 배포 파이프라인에 WIF 사용을 우선 권장하며, 필요한 IAM 역할들(예: Artifact Registry Writer, Cloud Run Admin, Service Account User, Workload Identity User)을 명시하고 있습니다 [1].

보안성 측면에서 WIF의 장점은 분명합니다. 첫째, **장기 액세스 키를 없애고** 매 실행 시마다 **단발성 토큰**을 사용하므로, 유출 위험을 크게 낮춥니다 [5]. 토큰은 짧게는 수분~1시간 내 만료되며(특히 ID 토큰의 유효기간은 최대 10분 정도입니다 [6]), 권한 남용 창구를 줄입니다. 둘째, 토큰에는 GitHub 워크플로우와 리포지토리 정보 등의 **컨텍스트가 포함**되어 GCP 측에서 해당 정보(예: `assertion.repository`, `assertion.ref`)를 검증할 수 있습니다 [7] [3]. 이를 통해 **신뢰 조건(Trust Conditions)**을 설정하면 특정 리포지토리 및 브랜치에서 오는 요청만 허용하는 등 세분화된 통제가 가능합니다. 이러한 **Zero Trust** 접근법은 보안성을 높여주며, 보고서의 방안은 이러한 조건부 접근을 전제로 한 것으로 추측됩니다.

확장성 및 한계도 함께 고려해야 합니다. WIF는 조직에 CI/CD 워크플로우가 늘어나더라도 **서비스 계정 키 배포 없이 확장**할 수 있다는 장점이 있습니다. 여러 저장소나 파이프라인에 대해 개별 키를 관리할 필요 없이, 해당 워크로드별로 WIF 프로바이더와 IAM 조건만 추가하면 됩니다. **관리 용이성** 면에서 이는 확장성에 유리합니다. 다만 **구성 복잡도**가 초기 진입장벽인데, 특히 다수의 리포지토리에 대해 세밀한 조건을 걸어야 할 경우 설정 작업이 번거로울 수 있습니다. 또한 WIF 토큰 교환은 GCP IAM의 **STS(Security Token Service)**를 거치므로 약간의 추가 지연이 생길 수 있지만, 일반적으로 몇백 ms 수준으로 CI/CD 파이프라인 전체에 큰 영향은 없습니다.

한계점으로는, **외부 IdP (예: GitHub)**에 전적으로 신뢰를 둔다는 점이 있습니다. GitHub의 OIDC 토큰 발급 메커니즘이 신뢰 기반이므로, GitHub 보안이 저해되면 WIF에도 영향이 있습니다. 그러나 업계 표준에 따라 잘 구현되어 있고, GCP 측에서도 발급자 URL과 JWT 서명 등을 철저히 검증하므로 현실적인 위험은 낮습니다. 결론적으로 WIF 방안은 **실**

용성과 보안성에서 모두 이점이 크지만, **초기 설정의 복잡성**과 **정책 관리의 복잡도**가 잠재적 한계로 지적될 수 있습니다.

3. Cloud Run 환경에서 서비스 계정 Impersonation/서비스 간 인증 대안

WIF는 주로 **외부 워크로드(GitHub 등)**의 GCP 인증에 쓰이는 방법입니다. **Cloud Run 서비스 간 통신이나 GCP 리소스 접근** 시에는 WIF 없이도 보다 간단한 **내부 인증 메커니즘**들을 활용할 수 있습니다. 몇 가지 실용적 대안을 제시하면 다음과 같습니다:

- **Cloud Run 간** 서버리스 IAM **인증**: Cloud Run 서비스 A가 서비스 B를 호출해야 하고 B가 **인증 필요**로 설정된 경우, **ID Token**을 사용한 **서비스 투 서비스 인증**이 가능합니다. Cloud Run에 할당된 서비스 계정이 있다면, 해당 워크로드 내에서 **메타데이터 서버**를 통해 자기 자신의 ID 토큰을 발급받을 수 있습니다 [8] . 예를 들어, Cloud Run 컨테이너 안에서 아래와 같이 metadata URL을 호출하면 자신을 나타내는 Google 서명 ID 토큰을 얻을 수 있습니다:

  ```
  curl -H "Metadata-Flavor: Google" \
    "http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/identity?audience=<TARGET_URL>"
  ```

  위와 같이 발급된 토큰을 **Authorization: Bearer** 헤더로 붙여 Cloud Run B를 호출하면, B는 해당 토큰을 통해 호출자가 **사전 승인된 GCP 서비스계정**임을 검증하고 요청을 수락합니다 [9] [10] . 이 방법은 Cloud Run 환경 내에서 **자동화 수준이 높고** 추가적인 외부 설정 없이도 동작합니다. GCP 공식 문서에서도 metadata 서버 또는 `google-auth-library`를 이용한 ID 토큰 획득 방식을 안내하고 있으며, **GitHub Actions + WIF**보다 Cloud Run 내부 호출엔 이 방식이 훨씬 간단합니다 [11] .

- **서비스 계정 Impersonation (위임)**: 만약 Cloud Run 서비스가 **다른 GCP 리소스**에 접근하는데, 현재 실행 서비스계정 권한보다 높은 권한이 일시적으로 필요하다면 **서비스계정 위임**을 고려할 수 있습니다. GCP IAM에서 소스 서비스계정에게 타겟 서비스계정을 **impersonate**할 수 있는 권한(`roles/iam.serviceAccountTokenCreator`)을 부여하면, 코드 상에서 단기 토큰을 발급받아 타겟 서비스계정으로 행세할 수 있습니다. 예를 들어 `source-sa`가 `target-sa`를 가장하도록 설정하려면 아래와 같은 IAM 바인딩을 추가합니다:

  ```
  gcloud iam service-accounts add-iam-policy-binding target-sa@PROJECT_ID.iam.gserviceaccount.com \
    --member="serviceAccount:source-sa@PROJECT_ID.iam.gserviceaccount.com" \
    --role="roles/iam.serviceAccountTokenCreator"
  ```

  그 후 Cloud Run의 `source-sa` 환경에서 Google Credentials API를 통해 `target-sa`의 액세스 토큰이나 ID 토큰을 얻어 사용할 수 있습니다. 이 방식은 **WIF 없이도 서비스계정 간 권한 위임**을 가능하게 하며, CI 파이프라인이 아닌 **런타임 서비스 간**에 적합한 패턴입니다. 다만 구현 난이도가 조금 있을 수 있으므로, **가능하면 Cloud Run 자체 서비스계정에 필요한 최소 권한을 직접 할당**하는 편이 단순합니다.

- **기타 대안**: 조직 내 CI/CD에 GitHub Actions 외에 Cloud Build 같은 **GCP 네이티브 CI**를 활용하는 것도 방법입니다. Cloud Build나 Cloud Composer 등 GCP 서비스는 기본적으로 GCP 리소스 접근에 자기 서비스계정을 활용하므로, 별도 WIF 설정 없이 IAM 권한만으로 안전하게 작업할 수 있습니다. 또한, Cloud Run 호출을 외부에서 해야 하는 시나리오에서는, Cloud Run을 **IAM 인증 필요**로 설정하고 **caller 서비스계정에 Invoker 역**

할을 주는 방식으로 보안성을 높일 수 있습니다 [12] [13]. 이 경우 외부 호출자는 사전에 신뢰된 GCP 서비스계정의 단기 토큰을 사용해야 하므로, 서비스 간 호출을 WIF 없이도 안전하게 제어할 수 있습니다.

요약하면, **Cloud Run 환경 내 자동화된 인증**에는 **메타데이터 기반 ID 토큰 발급**이나 **IAM을 통한 서비스계정 위임** 등이 WIF 대비 실용적입니다. 이런 방법들은 GCP가 제공하는 기본 인프라를 활용하므로 설정이 단순하고, CI 파이프라인 외에 **런타임 서비스 호출**에 최적화되어 있습니다.

4. 보고서 미언급 또는 간과된 보안 위협 요소 및 보완책

보고서에서 WIF 구성 자체에 집중하다 보면 몇 가지 **추가적인 보안 위협 요소**를 간과하기 쉽습니다. 아래에 그런 요소들과 대응 방안을 정리했습니다:

- **OIDC 토큰 탈취 및 오용**: WIF 환경에서는 GitHub 등이 발급한 OIDC **ID 토큰**이 GCP에 제출되어 액세스 권한을 얻습니다. 이 토큰이 중간에 유출될 경우, 만료 전까지 공격자가 GCP 리소스에 접근할 위험이 있습니다. 이를 막기 위해 **토큰의 유효시간을 최소화**하고(앞서 언급했듯 최대 10분 수준 [6]) **Audience 제한**을 엄격히 합니다. GitHub OIDC 토큰의 기본 aud는 Google WIF Provider 리소스에 한정되며, GCP STS 교환에만 쓰입니다 [7]. 따라서 유출 위험을 줄이려면 **토큰을 로그에 남기지 않고**, 교환 후 바로 폐기하며, 워크플로우 실행 후 환경 변수 등을 소거해야 합니다. 보완책으로 GCP 측 **Security Token Service 호출에 VPC Service Controls**를 적용해 외부에서 STS API에 접근 못하도록 차단하는 방법도 고려할 수 있습니다 (STS를 민감서비스로 간주).

- **ID 토큰 캐싱 및 재사용**: WIF나 Cloud Run간 호출에서 발급된 ID 토큰을 재사용하거나 장시간 캐싱하는 것은 바람직하지 않습니다. 토큰은 짧은 수명 뒤 만료되므로 캐싱 시 **인증 실패**나 **만료 토큰 재사용 위험**이 생깁니다. 특히 **클라이언트 측에서 토큰을 캐싱**했다가 누출될 경우 문제가 될 수 있습니다. 권장 보완책은 **항상 호출 직전에 신규 토큰을 발급**받는 것입니다. 예를 들어 Cloud Run->Cloud Run 통신도 요청마다 metadata 서버에서 fresh 토큰을 가져오도록 하고, GitHub Actions에서도 `google-github-actions/auth` 액션이 매 job마다 토큰을 교환하도록 합니다. 불가피하게 토큰을 저장해야 한다면 (예: 워크플로우 내 여러 단계에서 동일 토큰 사용) **GitHub Actions의 secure storage**나 **메모리 상 변수**로만 유지하고 디스크에 남기지 않습니다.

- **Workload Identity Federation 구성 오용**: 잘못된 WIF 설정은 보안 취약으로 이어질 수 있습니다. **Trust 조건을 너무 완화**하게 설정하면, 원래 의도하지 않은 외부 주체도 토큰을 받아 사용하게 될 수 있습니다. 예를 들어 특정 GitHub 리포지토리로 제한하지 않고 전체 GitHub (`attribute.repository` 미사용) 토큰을 허용하면, **아무 GitHub 워크플로우나** 우리 GCP에 접근할 수 있는 위험이 있습니다. 반드시 **속성 매핑과 조건**을 활용해 허용할 리포지토리/브랜치를 구체적으로 지정해야 합니다 [3] [4]. 상기한 바와 같이 GCP IAM 정책에 `principalSet://.../attribute.repository/<ORG>/<REPO>` 형식으로 **특정 저장소만 허용**하는 바인딩을 걸어두면, 그 외 토큰은 배제됩니다. 아래는 예시 IAM 구성입니다:

```
gcloud iam service-accounts add-iam-policy-binding "deployer-sa@PROJECT_ID.iam.gserviceaccount.com" \
  --role="roles/iam.workloadIdentityUser" \
  --member="principalSet://iam.googleapis.com/projects/PROJECT_NUMBER/locations/global/workloadIdentityPools/POOL_ID/attribute.repository/my-org/my-repo"
```

[4]

이처럼 조건을 설정함으로써 **WIF 오용 가능성을 최소화**해야 합니다. 아울러 **불필요한 권한 제거**(리스코프) 원칙에 따라 WIF로 연계된 서비스계정에는 꼭 필요한 IAM 역할만 주고 [14], 주기적으로 해당 역할을 검토해 과도한 권한이 부여되지 않도록 해야 합니다 [15].

- **외부 접근 경로 통제**: Cloud Run 서비스를 토큰 기반 인증으로 보호하더라도, 배포 설정에 따라 **공개 URL**로 노출되면 네트워크 레벨에서 누구나 접근 시도를 할 수 있습니다. 보고서에 없다면 고려해야 할 점은 **Cloud Run 자체 IAM Invoker 설정**입니다. 현재 Admin 토큰으로 애플리케이션 레벨 인증을 한다 해도, 가능하다면 Cloud Run 서비스 설정에서 **--allow-unauthenticated를 해제**하고 **Invoker를 제한**하는 것이 이중 보안에 좋습니다 [16] [17]. 다만 이 경우 프론트엔드나 호출 주체가 별도의 GCP **서비스계정 자격**으로 호출해야 하므로 구현 복잡도가 올라갈 수 있습니다. 조직의 보안 요구 수준에 따라 **네트워크 및 IAM 차원의 이중 잠금**도 검토하시기 바랍니다. 추가로, Cloud Run이 접근하는 DB나 외부 API 경로에 대해서도 **방화벽 및 VPC 설정**을 점검해, Cloud Run 외부에서 직접 접근 가능한 경로가 없도록 하는 것이 중요합니다 [17]. 예를 들어 Cloud Run이 Cloud SQL 등을 쓴다면 Cloud Run에 **VPC 커넥터**를 붙이고 DB 인스턴스를 사설 IP로 제한하는 등의 조치가 필요합니다.

정리하면, **보고서에 언급되지 않은 보완 포인트**로 (a) **토큰 유출 대비** – 짧은 수명과 로그 금지, (b) **토큰 캐싱 지양** – 매 요청 재발급, (c) **WIF 조건 엄격화** – 최소 권한의 원칙, (d) **이중 인증/네트워크 통제** – Cloud Run IAM 설정 등을 제시할 수 있습니다. 이러한 보완책을 적용하면 WIF 기반 인증의 안전성이 한층 강화될 것입니다.

5. 권장 인증 방식 우선순위 및 운영 정책

종합하면, 조직에서 Cloud Run 및 CI/CD 파이프라인 인증을 관리할 때 다음과 같은 **우선순위와 운영 원칙**을 권장합니다:

1. **GCP 내부 호출에는 기본 제공 인증 활용 우선**: Cloud Run, Cloud Functions 등 **GCP에서 실행되는 워크로드 간 통신**에는 가능하면 GCP의 기본 서비스계정 인증을 활용합니다. 별도의 키 발급 없이 **런타임 환경의 Application Default Credentials(ADC)**나 **메타데이터 ID 토큰**을 사용하여 인증하도록 설계합니다. 이를 위해 필요한 IAM 역할(예: Cloud Run Invoker 등)은 호출자 서비스계정에 부여하고, 대상 서비스는 IAM 인증을 요구하도록 설정합니다. 이 접근법은 **구성이 간단**하면서도 **자동으로 단기 토큰**을 사용하므로 보안성이 높습니다 [9] [10].

2. **외부 CI/CD 파이프라인에는 Workload Identity Federation 적극 활용**: GitHub Actions와 같이 GCP 외부에서 구동되는 파이프라인에는 **WIF를 1순위 인증 방식**으로 채택합니다. **서비스계정 키를 금지**하고, WIF 워크로드 풀 및 프로바이더를 통해 필요한 파이프라인에만 권한을 federate합니다. 운영 정책으로 각 프로젝트/저장소별로 WIF 구성을 문서화하고, 신규 파이프라인이 추가될 경우 **표준화된 WIF 설정 절차**에 따라 구성하도록 합니다. 예를 들어, GitHub 리포지토리를 온보딩할 때 `roles/iam.workloadIdentityUser` 바인딩과 GitHub Actions YAML에 `id-token: write` 권한 설정을 체크리스트에 포함시킵니다 [18] [19].

3. **서비스계정 키 최소화 및 관리**: 조직 정책으로 **서비스 계정 JSON 키 발급을 지양**하고, 가능한 한 WIF나 ADC로 대체합니다. 부득이 키를 사용해야 하는 레거시 워크로드가 있다면, **키 수명 주기 관리**(정기적 rotation)와 저장 위치 암호화, 사용범위 제한 등의 보안대책을 마련합니다. GCP Organization 정책으로 신규 키 발급을 제한하거나 (예: `iam.disableServiceAccountKeyCreation` 설정) 모니터링하여, 키가 발급될 경우 보안팀에서 검토하도록 합니다.

4. **최소 권한 원칙 및 세분화**: 인증 방식에 상관없이, **각 서비스/파이프라인에 꼭 필요한 권한만 부여**하는 원칙을 운영합니다 [14]. WIF의 federated SA에도 광범위한 권한 대신 특정 리소스 조작에 국한된 역할만 주고, Cloud Run 실행 서비스계정도 역할을 세분화합니다. 예컨대 Cloud Run 서비스가 Cloud Storage 읽기만 필요하다면 `roles/storage.objectViewer`만 부여하고 Editor 같은 포괄권한은 피합니다. 권한 변경이 발생하면 **Change Management 프로세스**를 통해 승인을 받고 문서에 남깁니다.

5. **정기 감사 및 모니터링**: 인증 시스템이 제대로 운용되고 있는지 **주기적으로 점검**합니다 [15]. WIF의 경우 Security Token Service 로그나 IAM 감사 로그를 검토하여 **예상치 못한 주체가 토큰을 교환**하지 않았는지 모니터링합니다. Cloud Run 호출의 경우 IAM 호출 로그에서 Unauthorized 시도가 없는지 확인합니다. 또한, 모든 파이프라인 및 서비스계정 권한을 **분기별로 리뷰**하여, 불필요해진 권한이나 사용되지 않는 설정(WIF 풀 등)을 제거합니다.

6. **교육과 가이드**: 마지막으로, 개발자와 운영팀에 **인증 모범사례**를 교육합니다. WIF 설정 방법, 토큰 취급 주의사항, Cloud Run 간 호출 시 인증 구현 방법 등을 내부 위키나 가이드로 제공하세요. 특히 **새로운 서비스나 파이프라인을 만들 때 어떤 인증 방식을 우선 쓸 것인지** 결정할 수 있는 흐름도를 만들어 두면 유용합니다 (예: "내부 서비스 간 -> Cloud IAM 사용, 외부 CI -> WIF, 제3자 앱 -> OAuth 사용자 인증" 등). 이를 통해 조직 전반에 일관된 인증 정책이 적용되도록 합니다.

위의 우선순위와 정책을 따르면, **키리스(keyless) 인증**과 **짧은 수명의 토큰 기반 접근**을 극대화하여 보안 수준을 높일 수 있습니다. 요약하면: **내부 통신에는 GCP 제공 인증을, 외부 파이프라인에는 WIF를 우선 적용**하고, 그 외 경우에도 가능하면 단기 자격증명 방식을 활용하는 것이 좋습니다. 이러한 전략 하에서 Cloud Run 환경을 운영하면, 인증 관리의 부담을 줄이면서도 안전하고 확장가능한 서비스를 유지할 수 있을 것입니다.

Sources: 주요 내용은 첨부 보고서의 내용을 토대로 하였으며, Google Cloud 공식 문서와 사례 블로그 [1] [5] [9] 등을 참고하여 일반적인 원칙과 구체적 예시를 보완하였습니다.

---

[1] Deploy to Cloud Run with GitHub Actions | Google Cloud Blog
https://cloud.google.com/blog/products/devops-sre/deploy-to-cloud-run-with-github-actions/

[2] [3] [4] [5] [7] [14] [15] Firefly | Setting Up Workload Identity Federation Between GitHub Actions and Google Cloud Platform
https://www.firefly.ai/academy/setting-up-workload-identity-federation-between-github-actions-and-google-cloud-platform

[6] GitHub - google-github-actions/auth: A GitHub Action for authenticating to Google Cloud.
https://github.com/google-github-actions/auth

[8] [9] [11] python 3.x - How do I get id_token to properly load in Cloud Run? - Stack Overflow
https://stackoverflow.com/questions/70038680/how-do-i-get-id-token-to-properly-load-in-cloud-run

[10] [12] [13] Authenticating service-to-service | Cloud Run | Google Cloud
https://cloud.google.com/run/docs/authenticating/service-to-service

[16] 1013_2QualiJournal 관리자 시스템 Cloud Run 배포 통합 가이드북.pdf
file://file-6FBNRBdWwrmEmYJ8dDgH26

[17] 1013_1QualiJournal 관리자 시스템 최종 통합 작업 가이드북.pdf
file://file-6DfuAdPY3q666RS3cV5ydM

[18] [19] Configure Workload Identity Federation with deployment pipelines | IAM Documentation | Google Cloud
https://cloud.google.com/iam/docs/workload-identity-federation-with-deployment-pipelines