Cross-app OAuth Attacks in Integration Platforms:

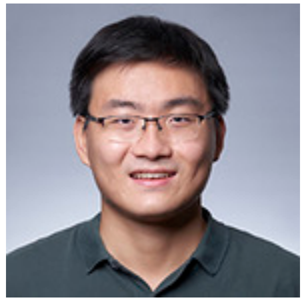
Mix-up Attacks Reloaded

Kaixuan Luo[1] **kaixuan@ie.cuhk.edu.hk**

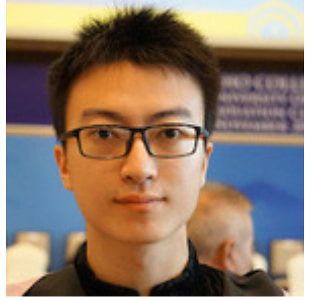
Xianbo Wang[1], Adonis Fung[2], Julien Lecomte[2], Wing Cheong Lau[1]

1 The Chinese University of Hong Kong, 2 Samsung Research America

About us



Kaixuan Luo*
PhD Candidate
kaixuan@ie.cuhk.edu.hk



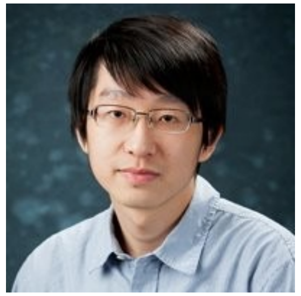
Xianbo Wang
PhD Candidate
🐦 @sanebow

Wing Cheong Lau
Professor

MobiTeC
Mobile Technologies Centre

香港中文大學
The Chinese University of Hong Kong

* Part of the work done while interning at Samsung



Adonis Fung
Director of Engineering, Security
Samsung Research America

Julien Lecomte
Head of Software Engineering & Operations
Samsung Research America

Samsung Research America

Agenda

Background: Integration Platform, OAuth Paradigm Shift

Highlights of our research: Cross-app OAuth Attacks

Suggested Changes: Based on OAuth Security BCP RFC

What are Integration Platforms?

By Usage Scenario

Workflow Automation Platforms

Microsoft Power Automate

IFTTT

Virtual Voice Assistants

Google Assistant

alexa

Smart Homes

Google Home

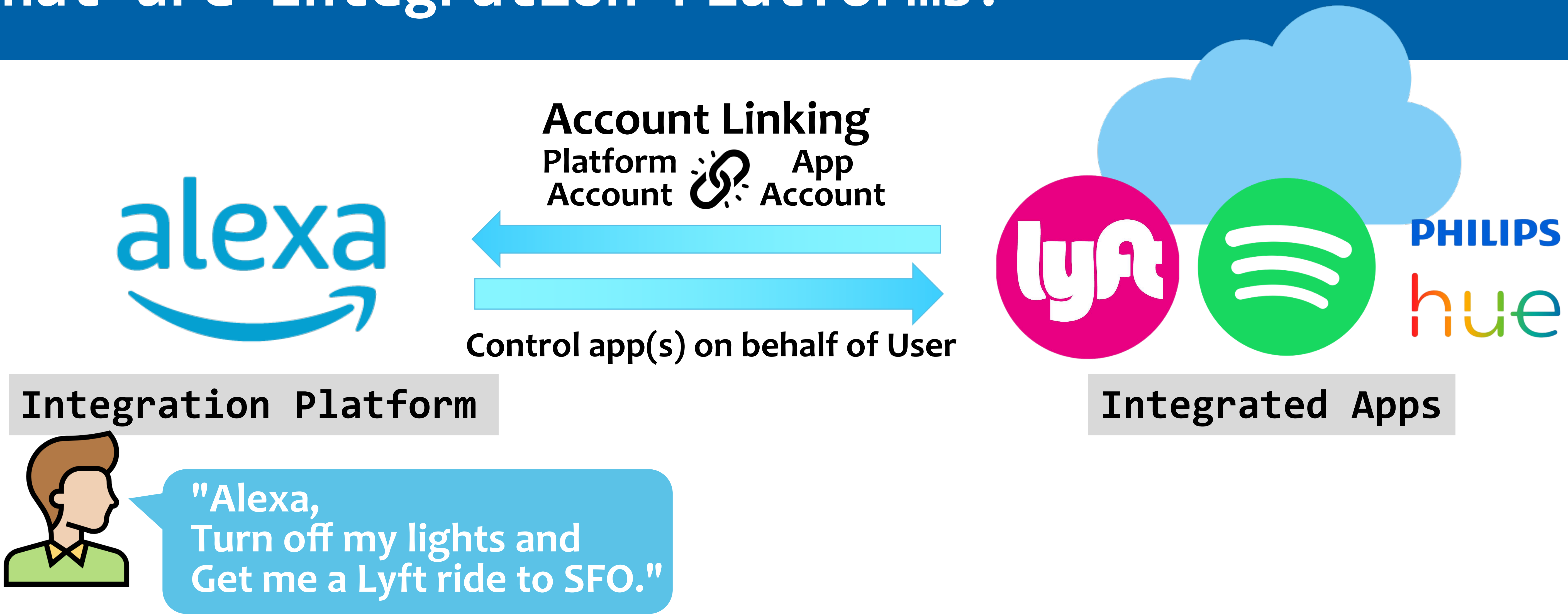
LLM Platforms with Plugins

By Development Approach

Trigger-action Platforms

Low-Code/No-Code Platforms

What are Integration Platforms?



Account Linking

Platform Account

App Account

Control app(s) on behalf of User

Integration Platform

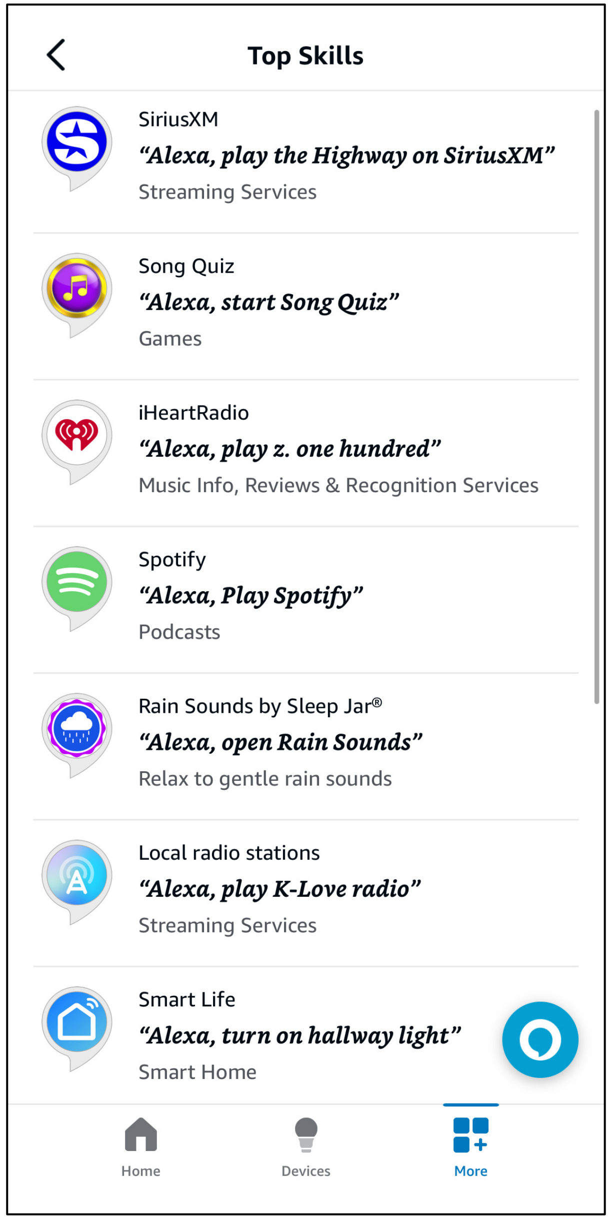
Integrated Apps

"Alexa,
Turn off my lights and
Get me a Lyft ride to SFO."

- **Integration Platform** Connects & Aggregates functionalities of diverse apps/ services/ devices
- **Account Linking** Links the end-user's App accounts to Integration Platform account
- **OAuth 2.0** is the de facto standard protocol to achieve Account Linking

Open Ecosystem: Marketplace Design

alexa



Top Skills

SiriusXM
"Alexa, play the Highway on SiriusXM"
Streaming Services

Song Quiz
"Alexa, start Song Quiz"
Games

iHeartRadio
"Alexa, play z. one hundred"
Music Info, Reviews & Recognition Services

Spotify
"Alexa, Play Spotify"
Podcasts

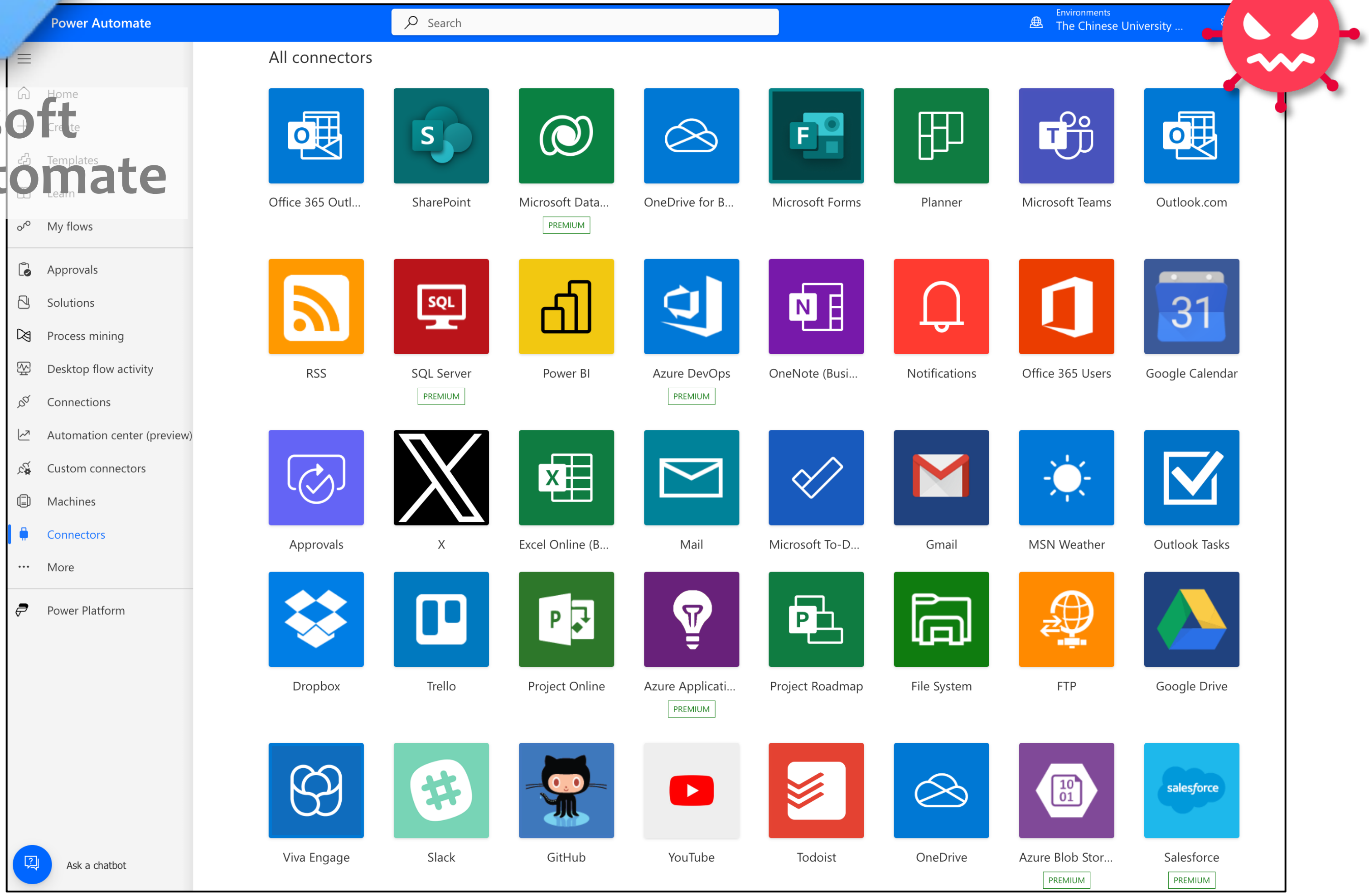
Rain Sounds by Sleep Jar®
"Alexa, open Rain Sounds"
Relax to gentle rain sounds

Local radio stations
"Alexa, play K-Love radio"
Streaming Services

Smart Life
"Alexa, turn on hallway light"
Smart Home

Microsoft
Power Automate

Anyone can publish an app



Power Automate

Search

Environments
The Chinese University ...

- Home
- Create
- Templates
- Learn
- My flows
- Approvals
- Solutions
- Process mining
- Desktop flow activity
- Connections
- Automation center (preview)
- Custom connectors
- Machines
- Connectors
- More
- Power Platform

Ask a chatbot

All connectors

Office 365 Outl...
SharePoint
Microsoft Data... PREMIUM
OneDrive for B...
Microsoft Forms
Planner
Microsoft Teams
Outlook.com

RSS
SQL Server PREMIUM
Power BI
Azure DevOps PREMIUM
OneNote (Busi...
Notifications
Office 365 Users
Google Calendar

Approvals
X
Excel Online (B...
Mail
Microsoft To-D...
Gmail
MSN Weather
Outlook Tasks

Dropbox
Trello
Project Online
Azure Applicati... PREMIUM
Project Roadmap
File System
FTP
Google Drive

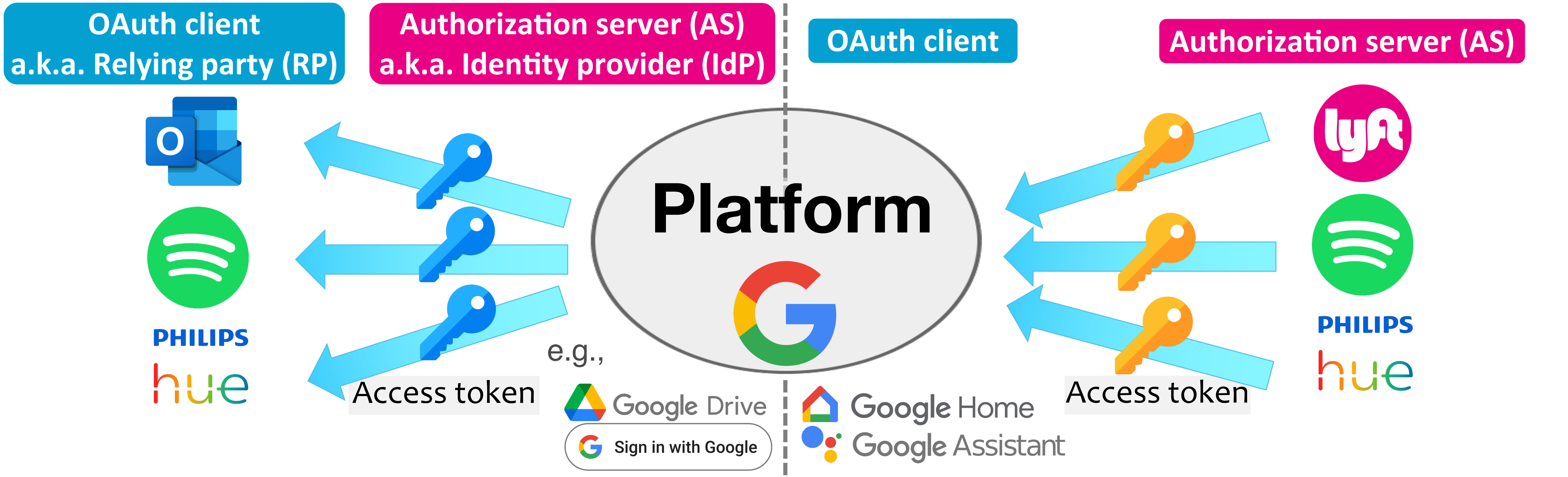
Viva Engage
Slack
GitHub
YouTube
Todoist
OneDrive
Azure Blob Stor... PREMIUM
Salesforce PREMIUM

Paradigm Shift: OAuth Role Reversal

End-user's Perspective

Traditional OAuth for Authorization or Single Sign-on (SSO)

⭐ OAuth for "Account Linking" in Integration Platforms



OAuth client
a.k.a. Relying party (RP)

Authorization server (AS)
a.k.a. Identity provider (IdP)

OAuth client

Authorization server (AS)

Platform

Access token

e.g.,

Google Drive

Sign in with Google

Google Home

Google Assistant

Access token

OAuth Apps

API/Identity Platform

Integration Platform

Integrated Apps

Potentially Untrusted

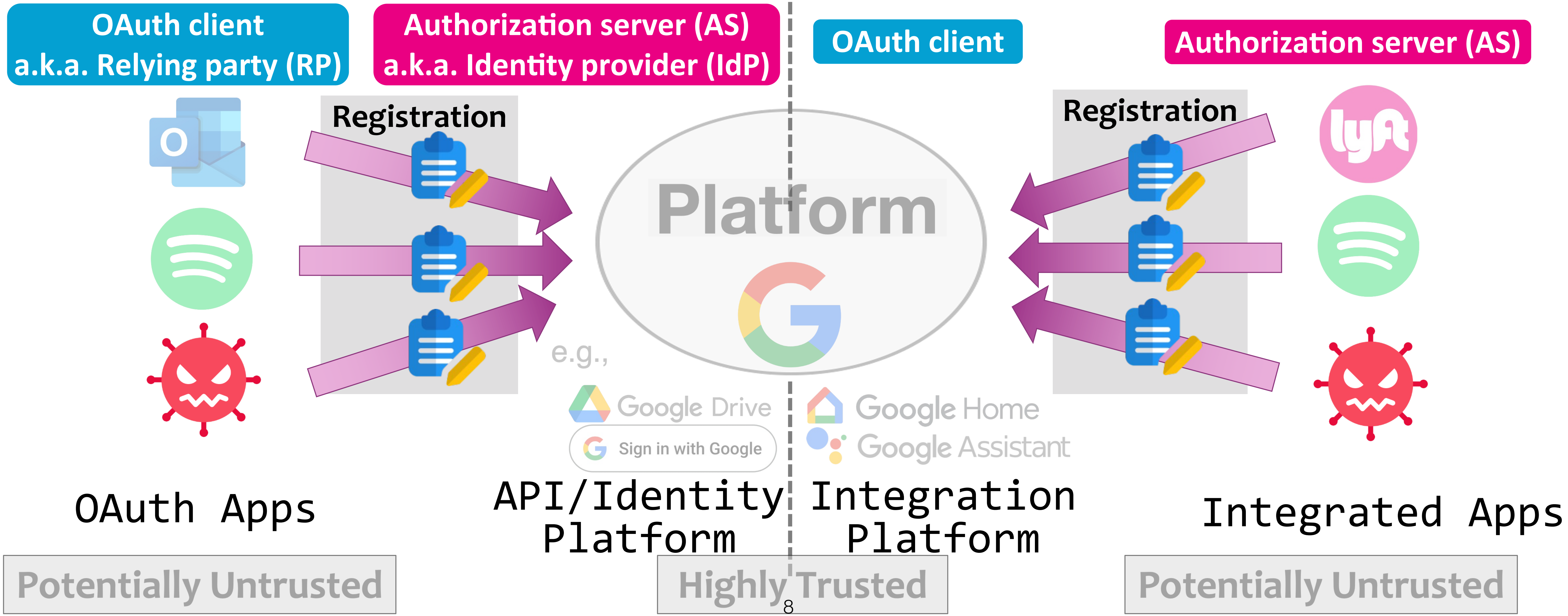
Highly Trusted

Potentially Untrusted

Paradigm Shift: OAuth Role Reversal
Developer's Perspective

Traditional OAuth for Authorization or Single Sign-on (SSO)

⭐ OAuth for "Account Linking" in Integration Platforms



OAuth client
a.k.a. Relying party (RP)

Authorization server (AS)
a.k.a. Identity provider (IdP)

Registration

Platform

e.g.,

Google Drive

Sign in with Google

OAuth Apps

API/Identity Platform

Potentially Untrusted

Highly Trusted

OAuth client

Authorization server (AS)

Registration

Google Home

Google Assistant

Integration Platform

Integrated Apps

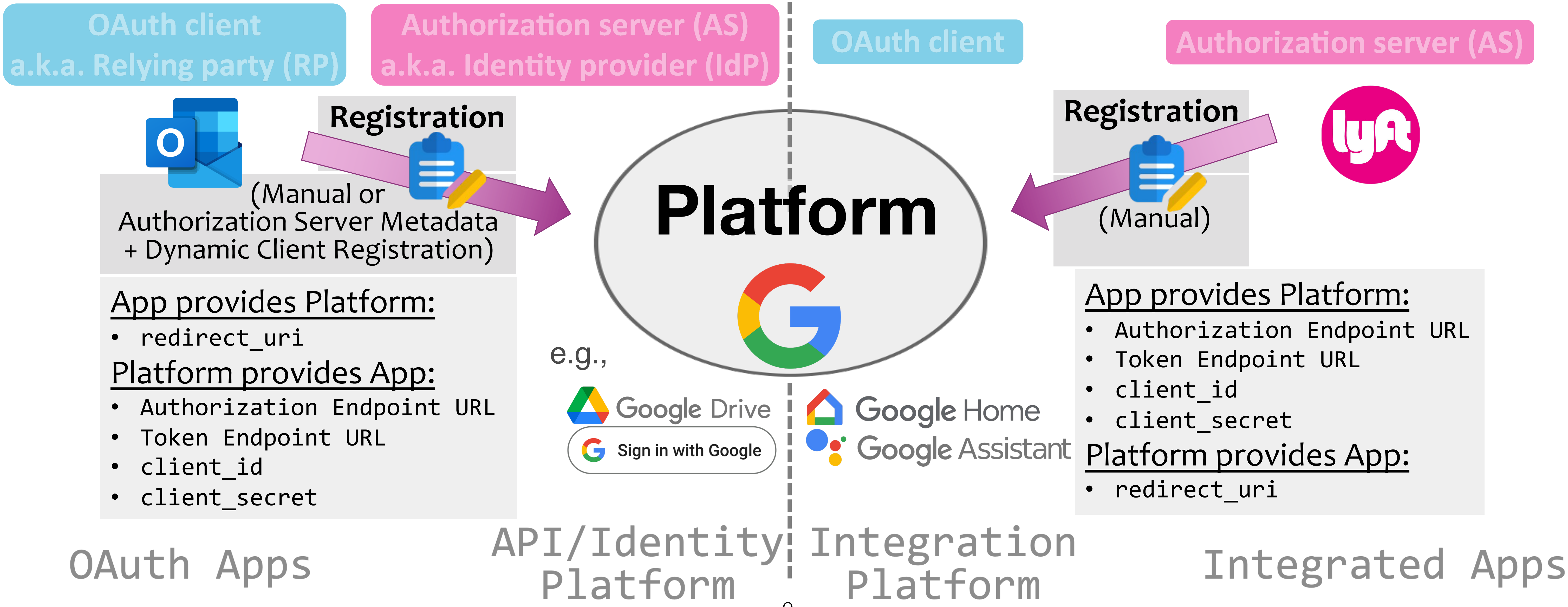
Potentially Untrusted

Paradigm Shift: OAuth Role Reversal

[Right-hand side]
Apps supply potentially malicious Auth/Token EPs; Platform supplies redirect_uri for each app

Traditional OAuth for Authorization or Single Sign-on (SSO)

⭐ OAuth for "Account Linking" in Integration Platforms



OAuth client a.k.a. Relying party (RP)

Authorization server (AS) a.k.a. Identity provider (IdP)

Registration

(Manual or Authorization Server Metadata + Dynamic Client Registration)

App provides Platform:
- redirect_uri

Platform provides App:
- Authorization Endpoint URL
- Token Endpoint URL
- client_id
- client_secret

OAuth client

Authorization server (AS)

Registration

(Manual)

App provides Platform:
- Authorization Endpoint URL
- Token Endpoint URL
- client_id
- client_secret

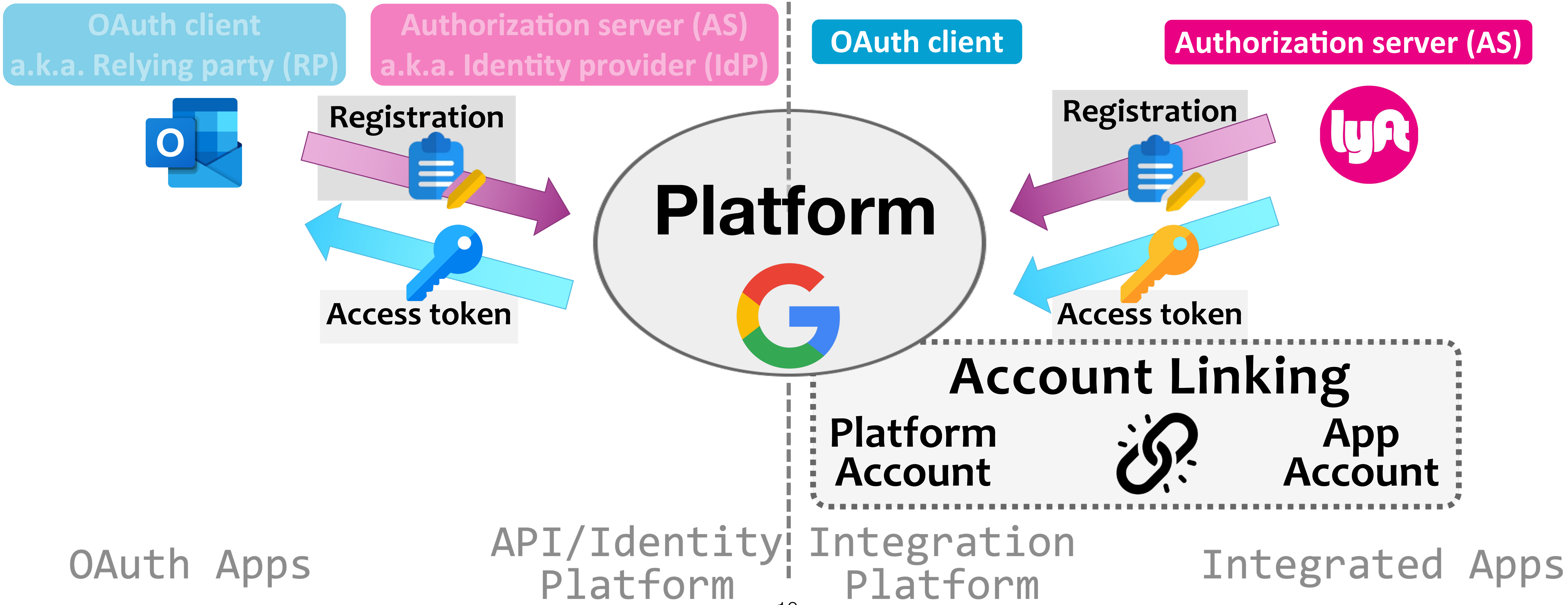
Platform provides App:
- redirect_uri

Paradigm Shift: OAuth Role Reversal

Full Picture

Traditional OAuth for Authorization or Single Sign-on (SSO)

OAuth for "Account Linking" in Integration Platforms



OAuth client
a.k.a. Relying party (RP)

Authorization server (AS)
a.k.a. Identity provider (IdP)

Registration

Access token

Platform

OAuth client

Authorization server (AS)

Registration

Access token

Account Linking

Platform Account

App Account

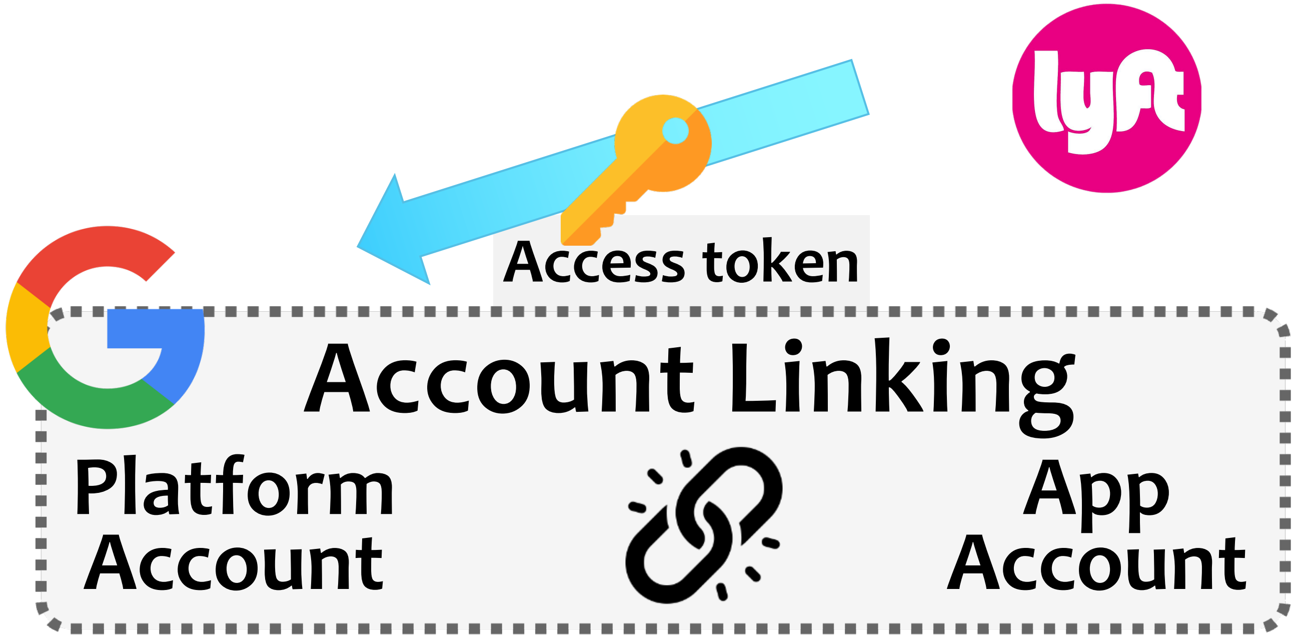
OAuth Apps

API/Identity Platform

Integration Platform

Integrated Apps

When OAuth-based Account Linking Goes Wrong



LGTM !	Users control their own apps	User	Same User
Unauthorized Access	Account Takeovers	Attacker	Victim
Privacy Leakage	Forced Account Linking	Victim	Attacker

How to Accomplish Goals

Account Linking

Platform Account	App Account
Attacker	Victim
Victim	Attacker

Unauthorized Access

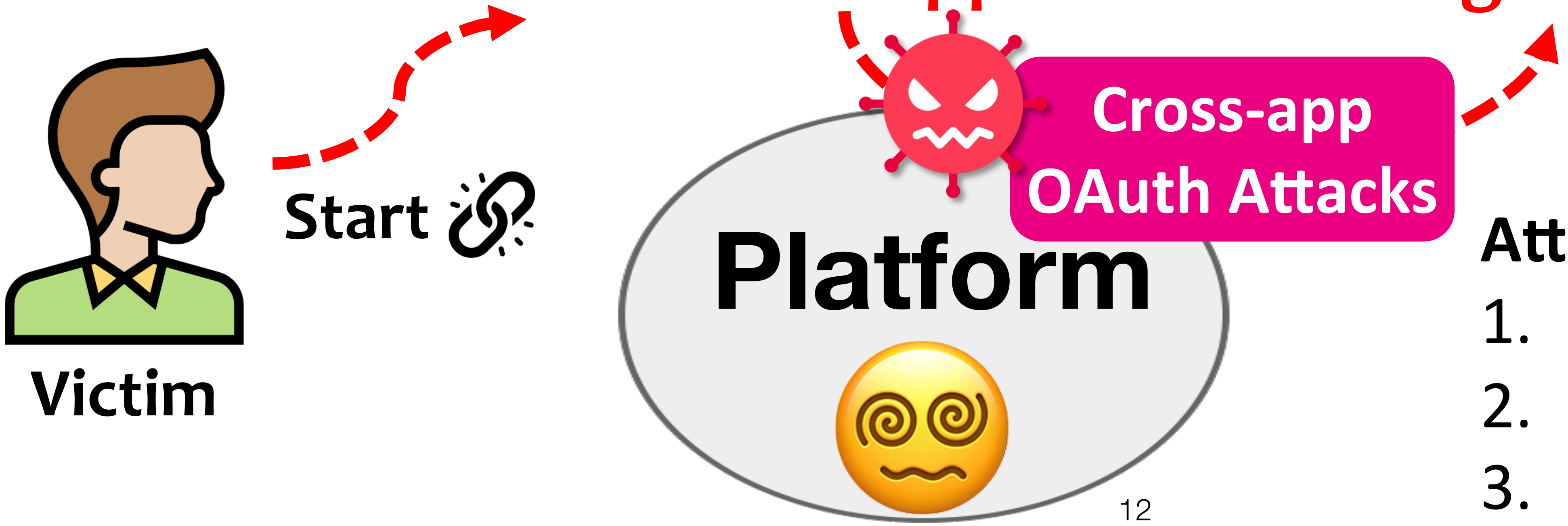
Account Takeovers

Privacy Leakage

Forced Account Linking

Attacker as a **Malicious App**

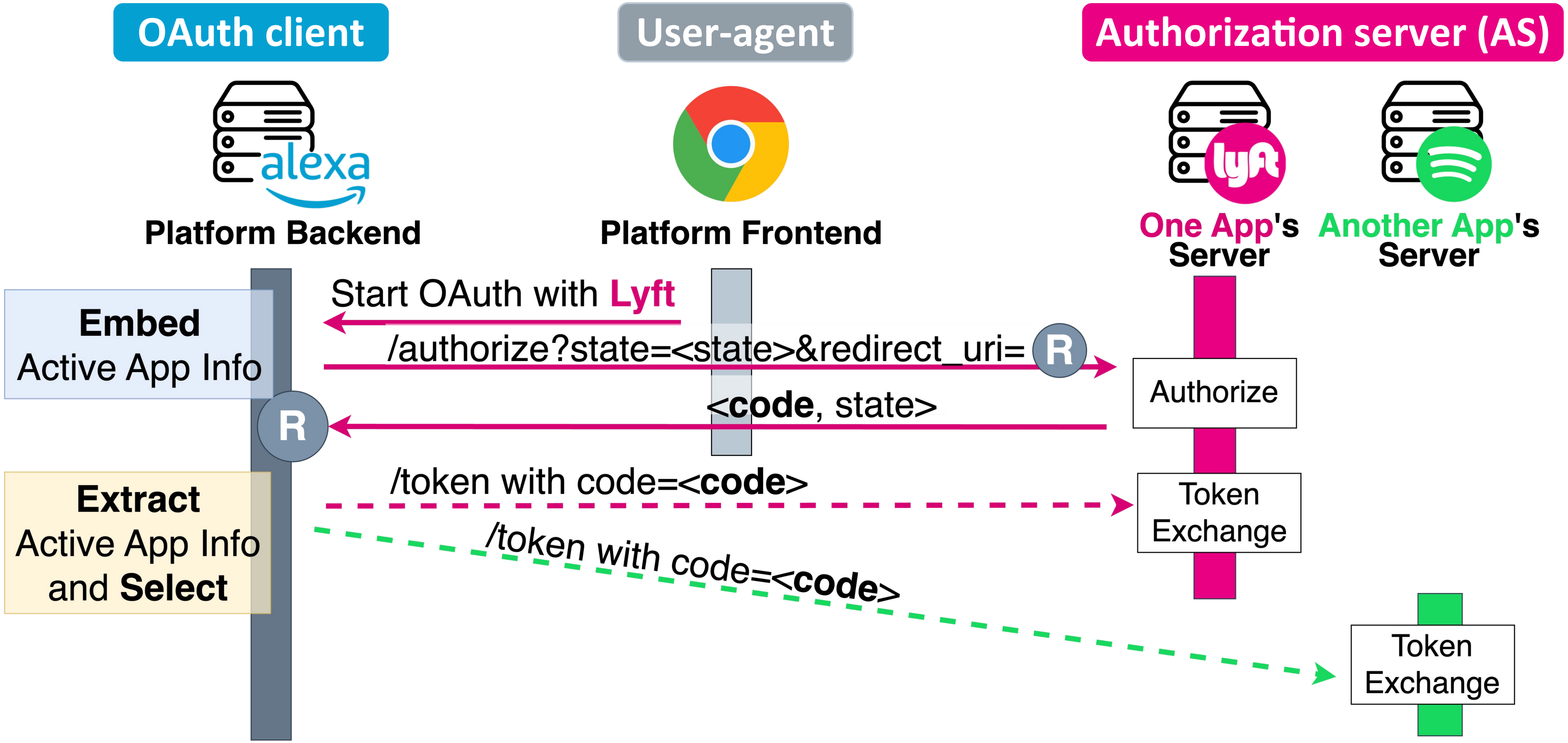
Victim's **Benign App** Account (Target)



Attack Scenario:

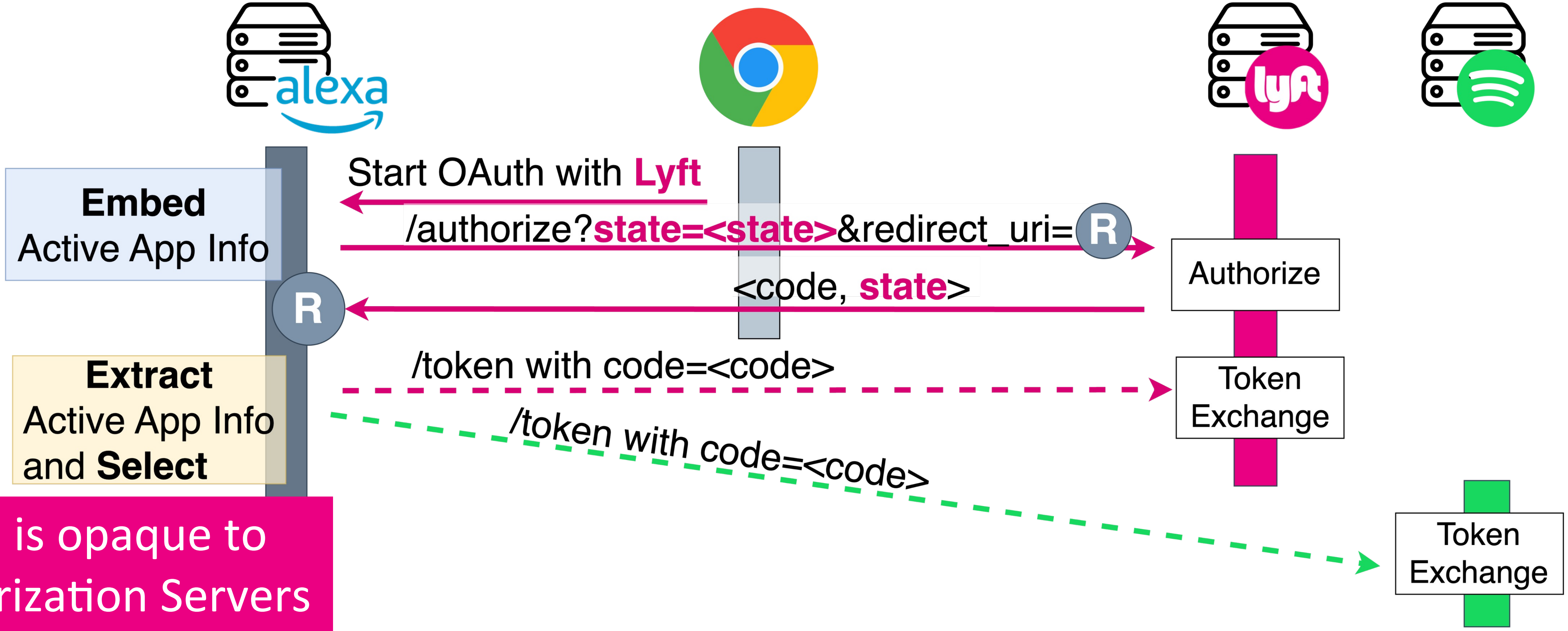
1. SET UP malicious app
2. TRICK the victim
3. CONFUSE the platform

Challenge: Supporting Multiple Integrated Apps



Active App Tracking

Common (but Flawed) designs for Active App Tracking

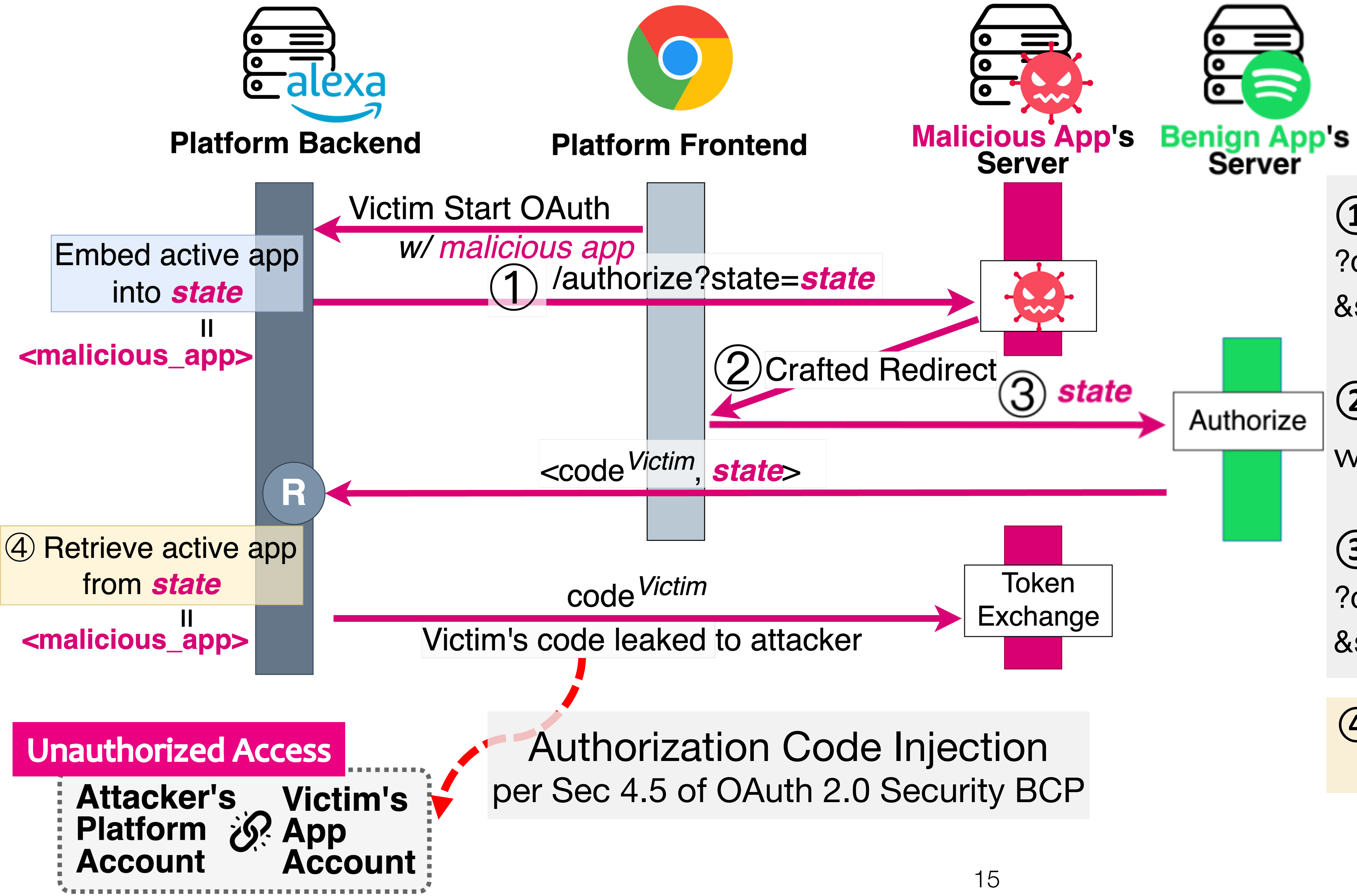


state is opaque to Authorization Servers

Platform shall embed in (and extract from):

- **state=eyJxxx.yyy.zzz**
  {"app_id": **<lyft>**,
  …}
  / platform's internal state
  (e.g., session, frontend-managed state)

Attack #1:
Cross-app OAuth Account Takeover (COAT)



Authorization Code Injection
per Sec 4.5 of OAuth 2.0 Security BCP

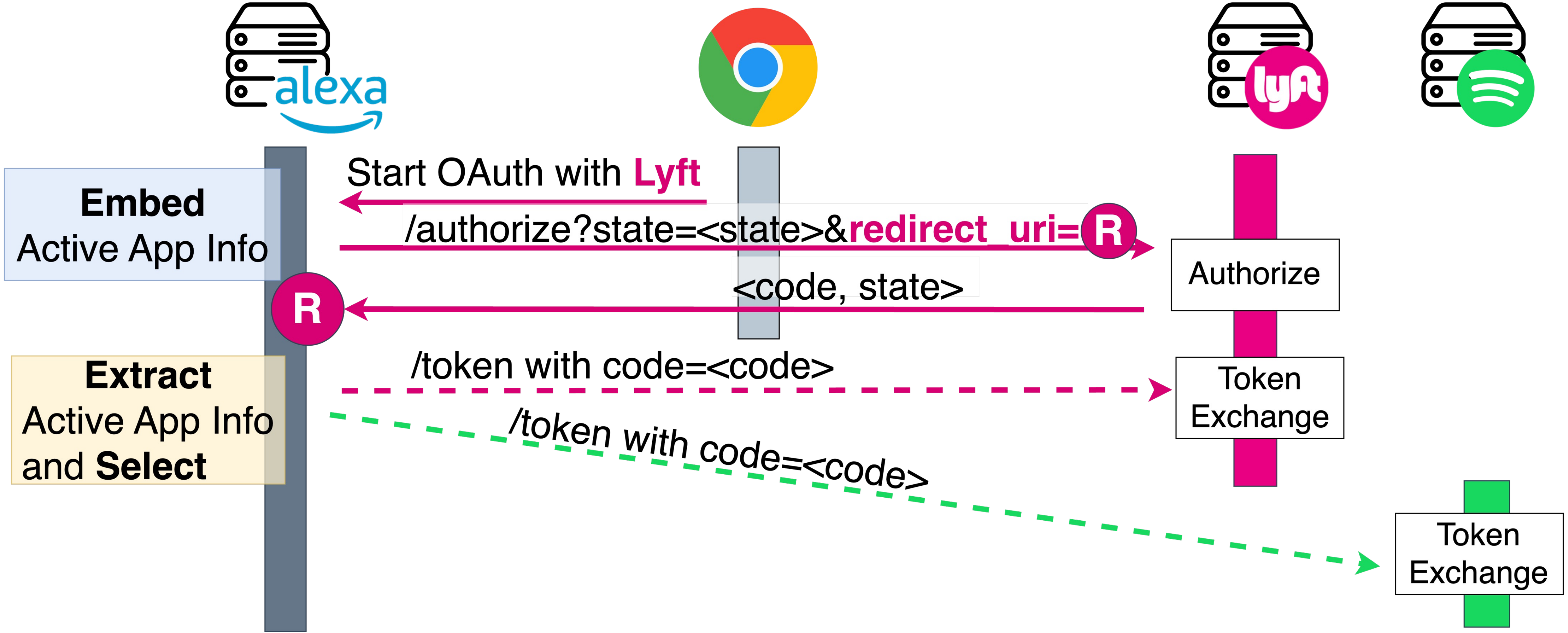
① **GET** https://**malicious.com**/authorize
?client_id+redirect_uri=**<malicious_app>**
&state=**<malicious_app>**

② Redirect to **benign app**, while keeping **malicious app**'s state

③ **GET** https://**benign.com**/authorize
?client_id+redirect_uri=**<benign_app>**
&state=**<malicious_app>**

④ **Flaw:** Track active app solely by `state`

Common (but Flawed) designs for Active App Tracking



Platform shall embed in (and extract from):

- state=eyJxxx.yyy.zzz
  {"app_id": <lyft>,
  …}
  / platform's internal state
  (e.g., session, frontend-managed state)

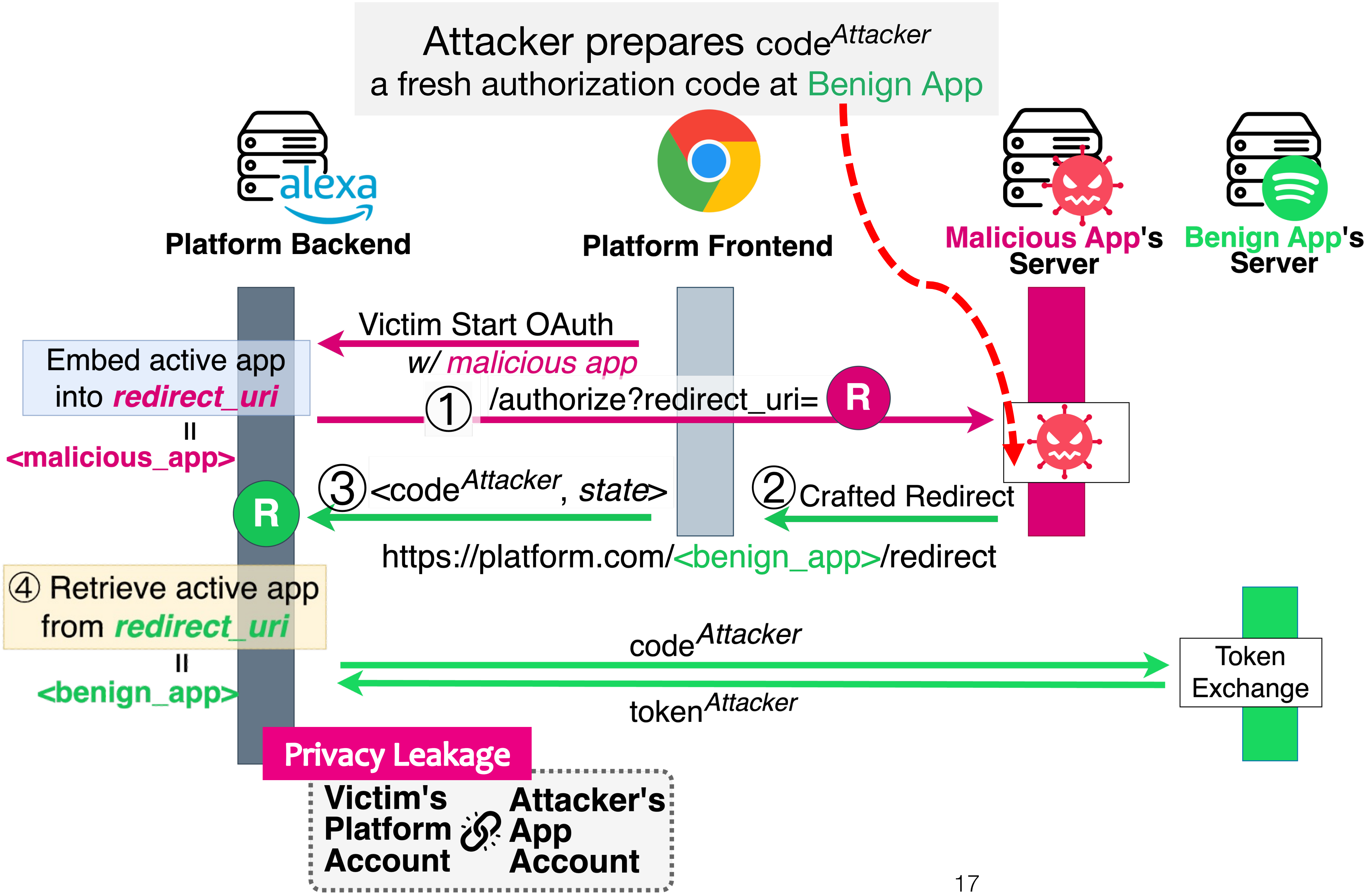
OR

- **redirect_uri:**
  https://platform.com/<lyft>/redirect

  app_id

redirect_uri has weak integrity

Attack #2:
Cross-app OAuth Request Forgery (CORF)



Attacker prepares code^{Attacker}
a fresh authorization code at Benign App

Platform Backend

Platform Frontend

Malicious App's Server

Benign App's Server

Victim Start OAuth *w/ malicious app*

Embed active app into ***redirect_uri***
II
<malicious_app>

① /authorize?redirect_uri= R

② Crafted Redirect

③ <code^{Attacker}, *state*>

https://platform.com/<benign_app>/redirect

④ Retrieve active app from ***redirect_uri***
II
<benign_app>

code^{Attacker}

token^{Attacker}

Token Exchange

Privacy Leakage

Victim's Platform Account **Attacker's App Account**

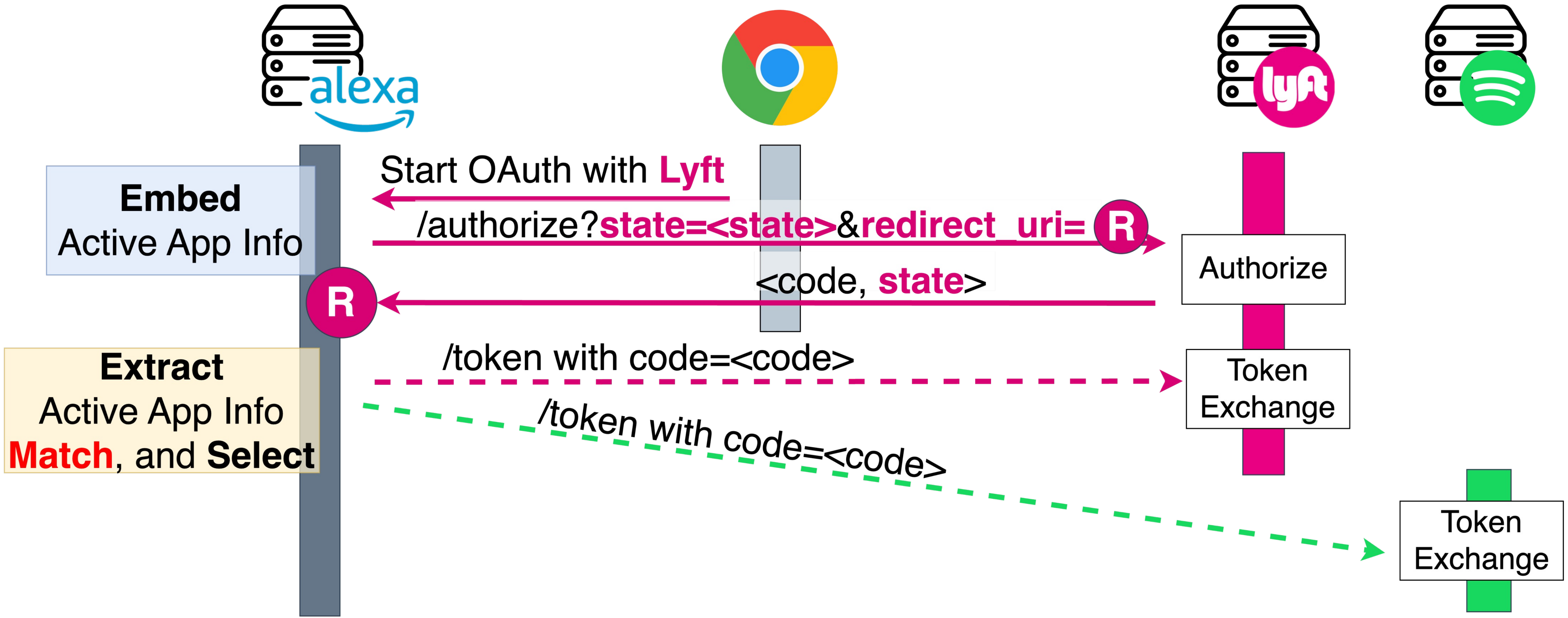
① **GET**
https://**malicious.com**/authorize
?client_id=**<malicious_app>**
&redirect_uri=platform.com/**<malicious_app>**/redirect&state=<state>

② Redirect to **benign app**'s R, Injecting the prepared code

③ **GET**
https://platform.com/**<benign_app>**
/redirect
?code=<attacker>&state=<state>

④ **Flaw:** Track active app solely by distinct **redirect_uri**

Defense for both COAT and CORF:
Consistency Check at redirection endpoint



①Shall embed **a unique app ID** in (and extract from) **BOTH**:

②Enforce **Matching** at R

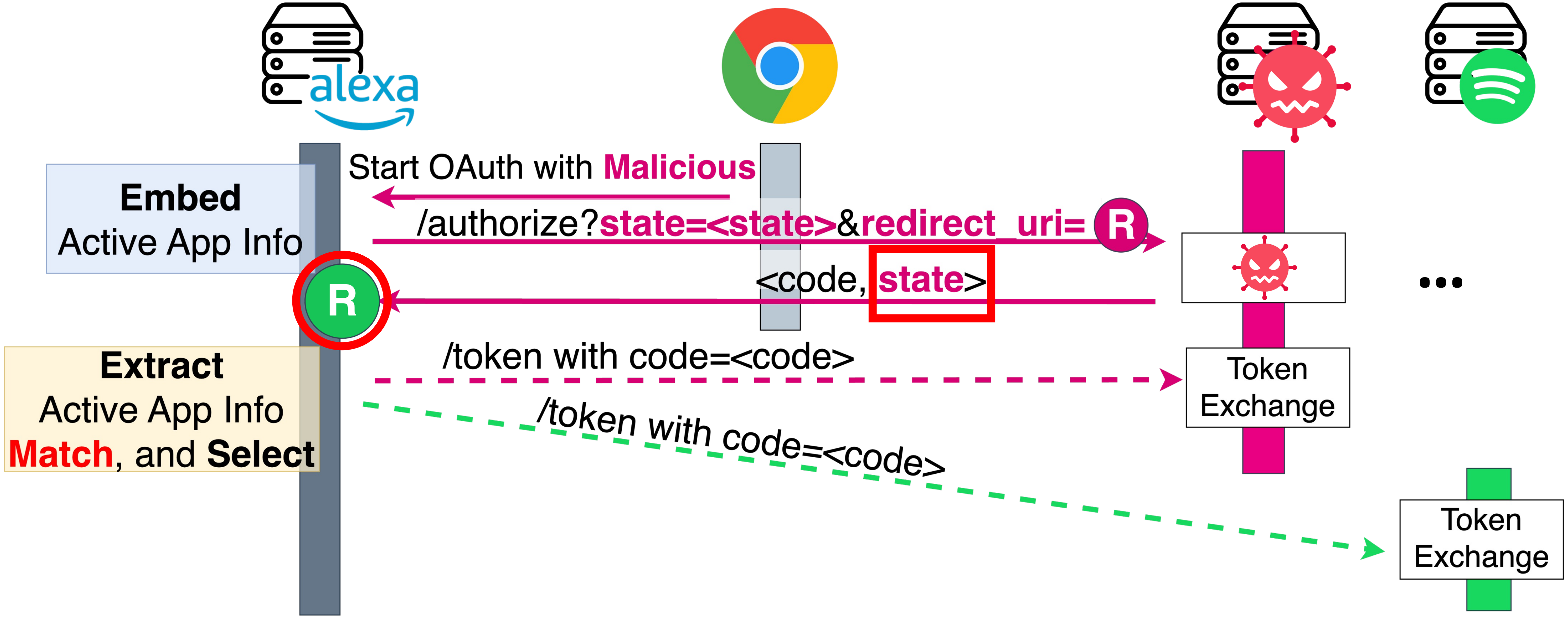
- state=eyJxxx.yyy.zzz
  {"app_id": <lyft>,
  …}
  / platform's internal state
  (e.g., session, frontend-managed state)

AND

- redirect_uri:
  https://platform.com/<lyft>/redirect

app_id

Defense for both COAT and CORF:
Why does it work?



Enforce Matching at R

- state=eyJxxx.yyy.zzz
  {"app_id": <malicious>,
  …}
  / platform's internal state
  (e.g., session, frontend-managed state)

BUT

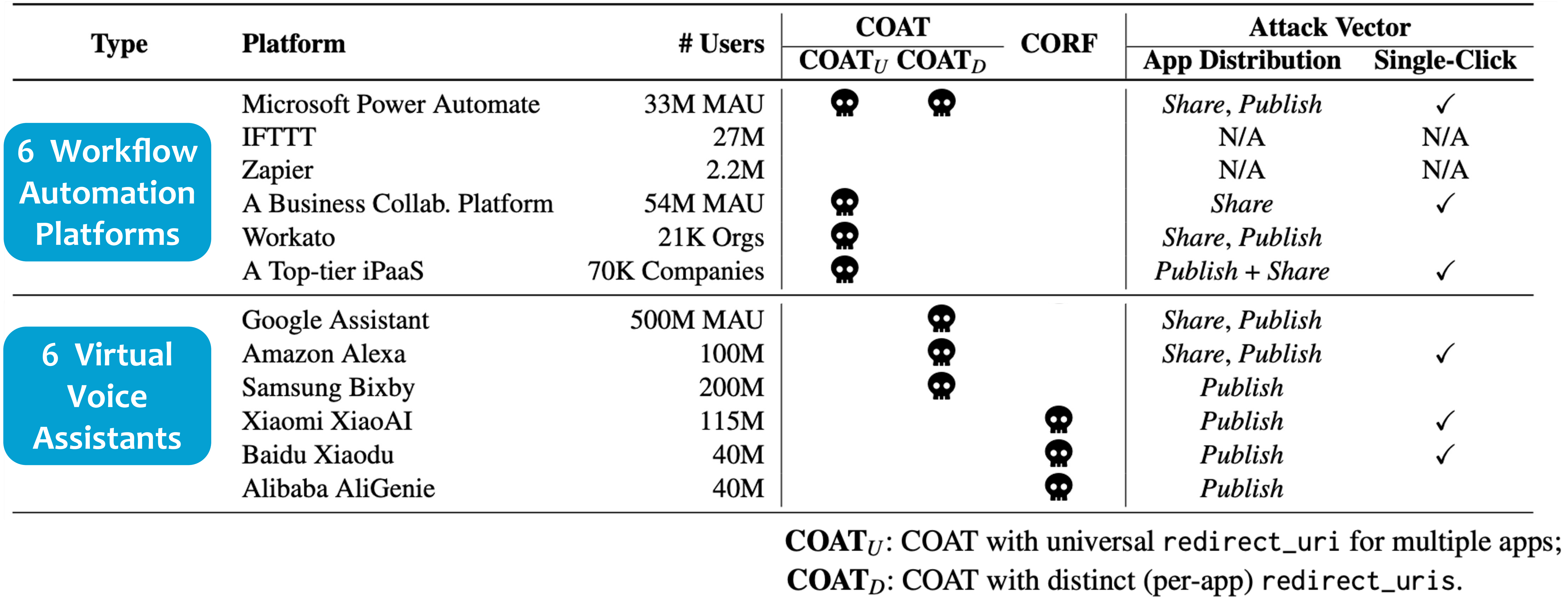
- redirect_uri:
  https://platform.com/<spotify>/redirect

app_id

Mismatch Detected!

Make the World a Better Place

Bug Hunting

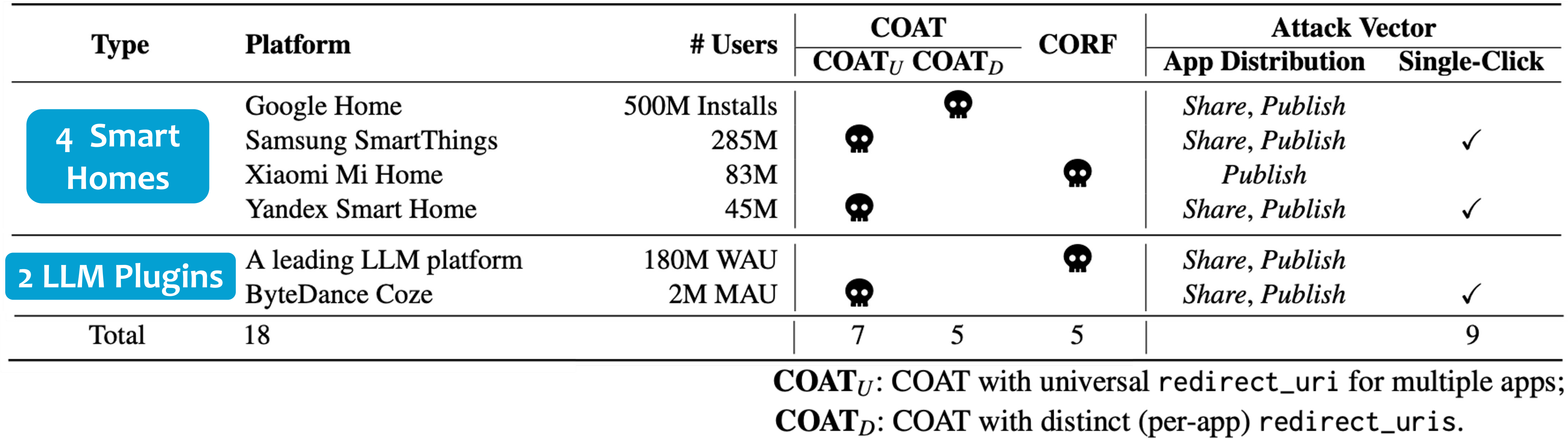


Type	Platform	# Users	COAT		CORF	Attack Vector	
			COAT$_U$	COAT$_D$		App Distribution	Single-Click
6 Workflow Automation Platforms	Microsoft Power Automate	33M MAU	☠	☠		*Share*, *Publish*	✓
	IFTTT	27M				N/A	N/A
	Zapier	2.2M				N/A	N/A
	A Business Collab. Platform	54M MAU	☠			*Share*	✓
	Workato	21K Orgs	☠			*Share*, *Publish*	
	A Top-tier iPaaS	70K Companies	☠			*Publish + Share*	✓
6 Virtual Voice Assistants	Google Assistant	500M MAU		☠		*Share*, *Publish*	
	Amazon Alexa	100M		☠		*Share*, *Publish*	✓
	Samsung Bixby	200M		☠		*Publish*	
	Xiaomi XiaoAI	115M			☠	*Publish*	✓
	Baidu Xiaodu	40M			☠	*Publish*	✓
	Alibaba AliGenie	40M			☠	*Publish*	

COAT$_U$: COAT with universal `redirect_uri` for multiple apps;
COAT$_D$: COAT with distinct (per-app) `redirect_uris`.

Make the World a Better Place

Bug Hunting (cont'd)



Type	Platform	# Users	COAT		CORF	Attack Vector	
			COAT$_U$	COAT$_D$		App Distribution	Single-Click
4 Smart Homes	Google Home	500M Installs		☠		*Share*, *Publish*	
	Samsung SmartThings	285M	☠			*Share*, *Publish*	✓
	Xiaomi Mi Home	83M			☠	*Publish*	
	Yandex Smart Home	45M	☠			*Share*, *Publish*	✓
2 LLM Plugins	A leading LLM platform	180M WAU			☠	*Share*, *Publish*	
	ByteDance Coze	2M MAU	☠			*Share*, *Publish*	✓
Total	18		7	5	5		9

COAT$_U$: COAT with universal `redirect_uri` for multiple apps;
COAT$_D$: COAT with distinct (per-app) `redirect_uris`.

Summary

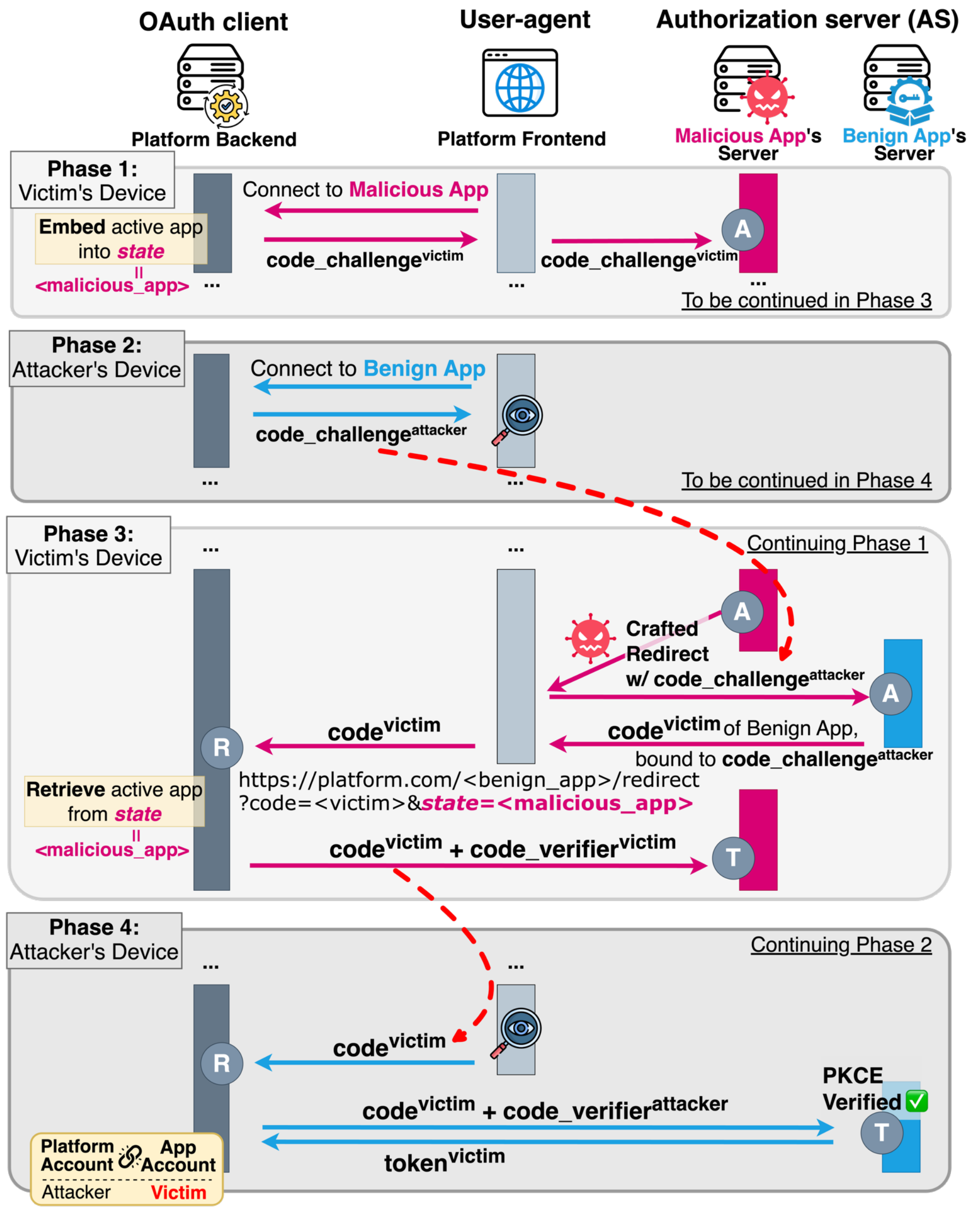
- **16/18** are vulnerable ☠
- **11** to COAT, **5** to CORF
- **9** can be done in 1-Click

Responsible Disclosure

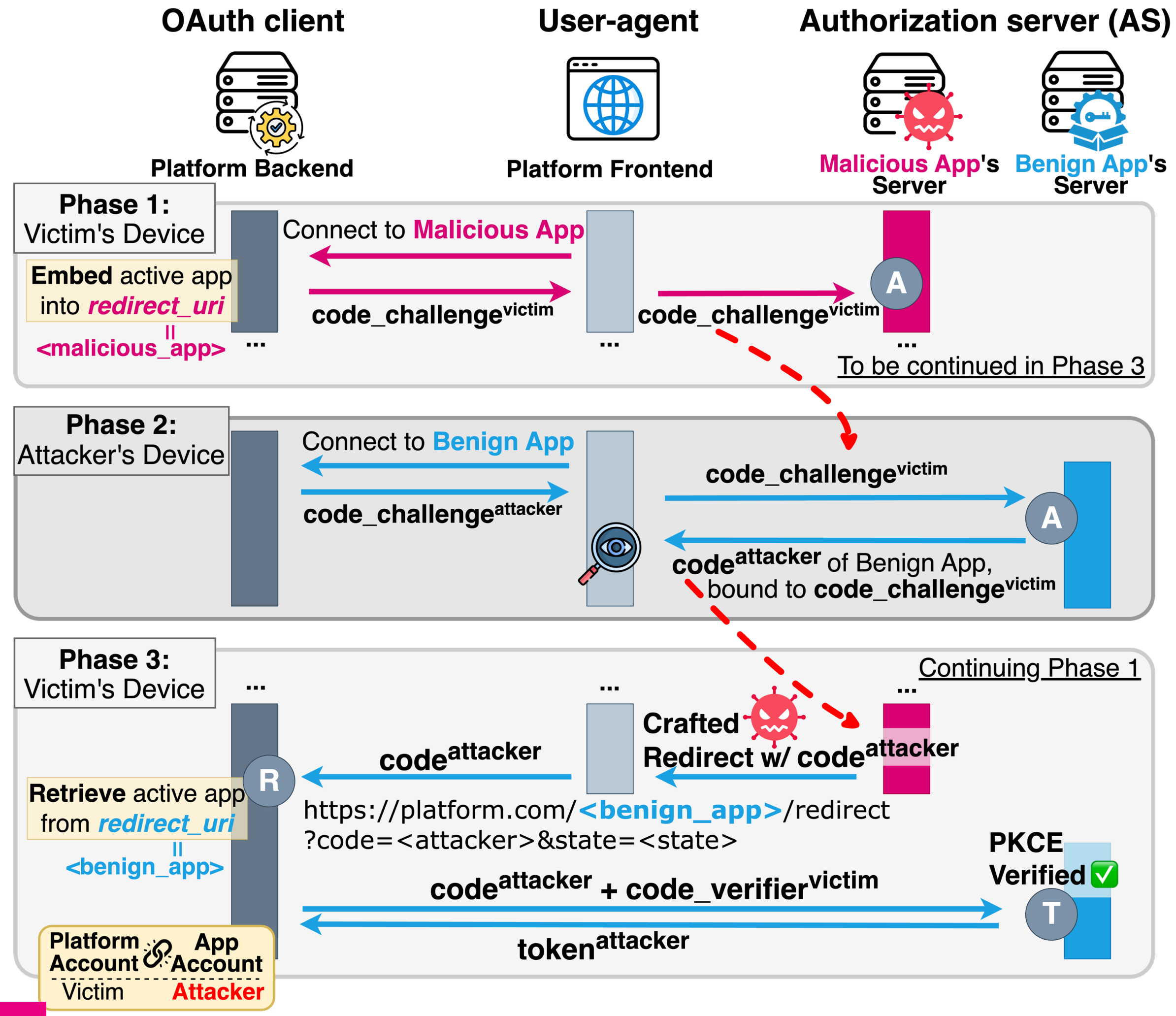
CVE-2023-36019 CVSS: 9.6

- Informed all 16 vulnerable platforms
- Confirmed by 11 platforms
- Patched by 9: 6 Robust fix, 3 Extra consent screen

FAQ 1: Isn't PKCE supposed to solve the problem?



OAuth client
User-agent
Authorization server (AS)
Platform Backend
Platform Frontend
Malicious App's Server
Benign App's Server
Phase 1: Victim's Device
Connect to Malicious App
Embed active app into state
<malicious_app>
code_challenge^victim
code_challenge^victim
A
To be continued in Phase 3
Phase 2: Attacker's Device
Connect to Benign App
code_challenge^attacker
To be continued in Phase 4
Phase 3: Victim's Device
Continuing Phase 1
Crafted Redirect w/ code_challenge^attacker
A
A
code^victim
code^victim of Benign App, bound to code_challenge^attacker
R
https://platform.com/<benign_app>/redirect ?code=<victim>&state=<malicious_app>
Retrieve active app from state
<malicious_app>
code^victim + code_verifier^victim
T
Phase 4: Attacker's Device
Continuing Phase 2
code^victim
R
code^victim + code_verifier^attacker
PKCE Verified ✅
T
token^victim
Platform Account App Account
Attacker Victim



Invalidate PKCE protection!

COAT w/ PKCE: (PKCE Chosen Challenge Attack)
Victim uses Attacker's code_challenge

CORF w/ PKCE:
Attacker uses Victim's code_challenge

FAQ 2: Isn't it mix-up attack?
Review on Mix-up Attacks in OAuth

Initial Discoveries

- [CCS 16] A Comprehensive Formal Security Analysis of OAuth 2.0
- [EuroS&P 17] SoK: Single Sign-On Security — An Evaluation of OpenID Connect

IdP Mix-up Attack

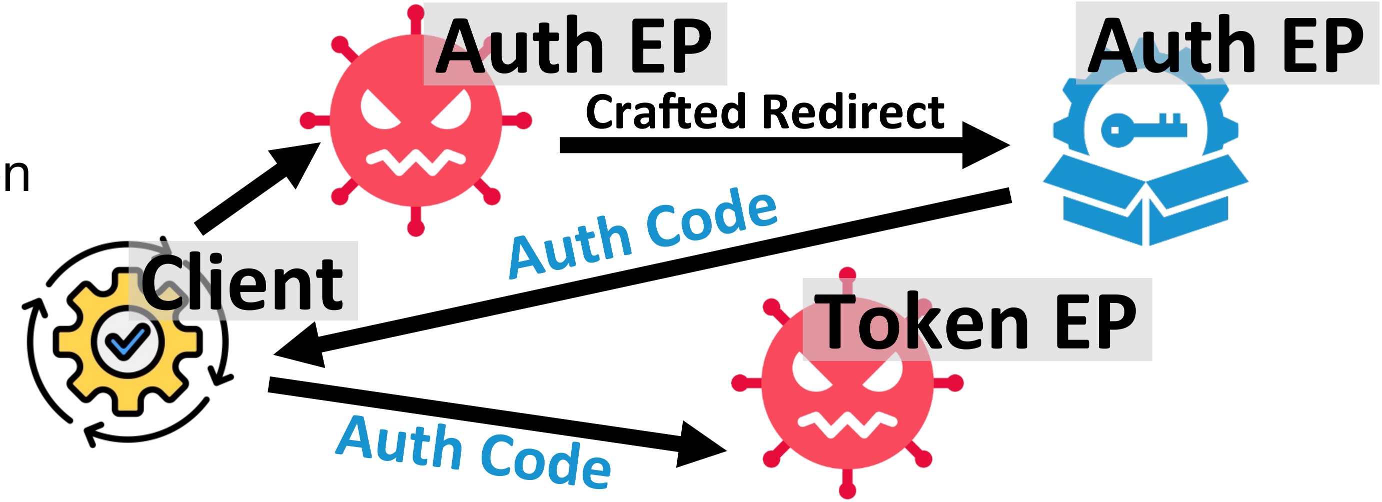
IdP Confusion /
Malicious Endpoints Attack

OAuth Security Workshop (OSW) Sessions

- [OSW 15] Initial Discussions by Daniel Fett, Christian Mainka et al. [summary]
- [OSW 16] "Does the IdP Mix-Up attack really work?" by Wanpeng Li [slides] [whitepaper]
- [OSW 16] "OAuth 2.0 Mix-Up Mitigation: Status and Next Steps" by Michael B. Jones [proposal]
- [OSW 21] "Overall pictures of Identity provider mix-up attack patterns and trade-offs between costs and effects for its mitigations" by Yoshiyuki Tabata [slides] [video]

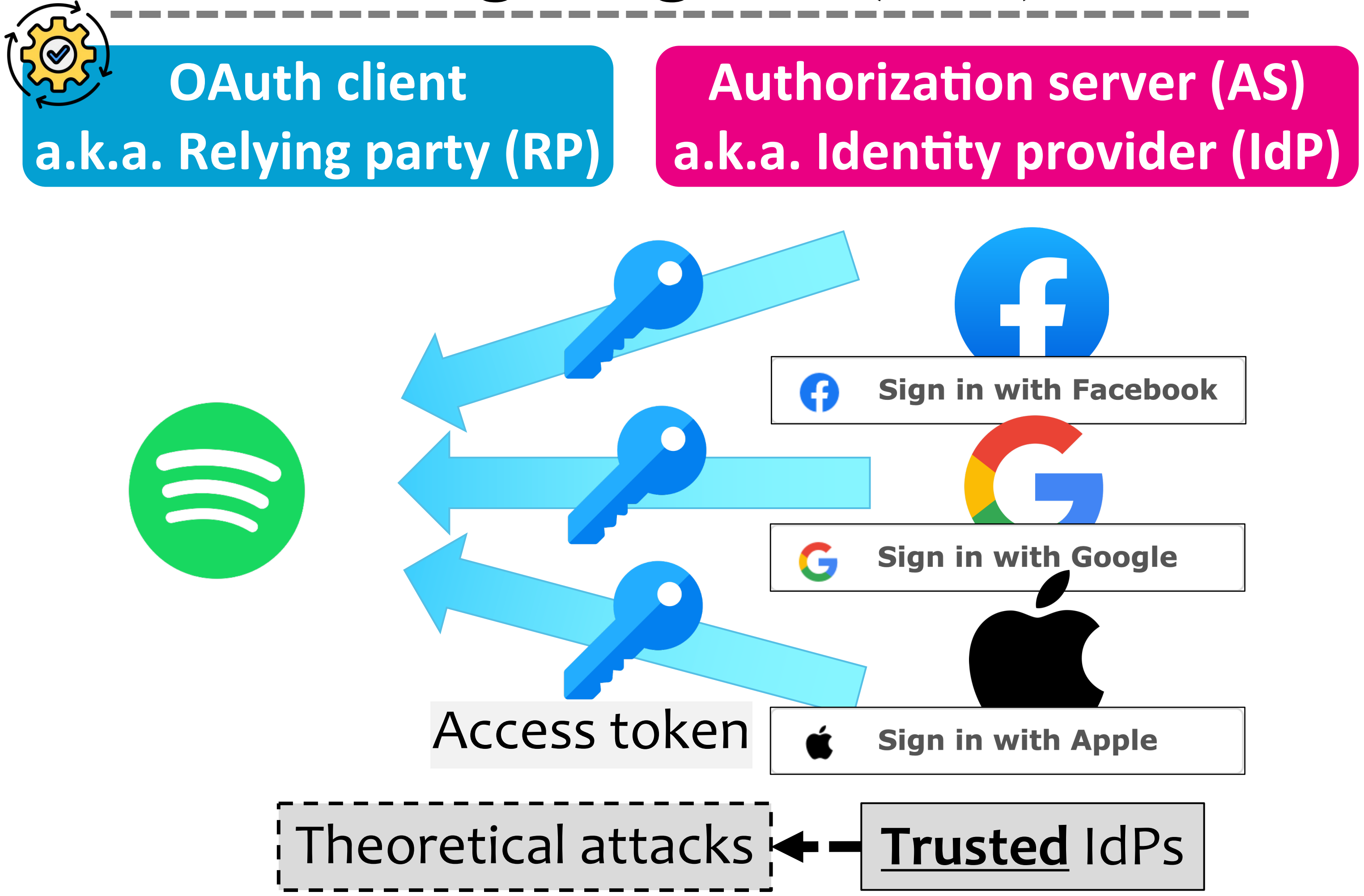
Standardization Efforts

- [RFC9207] OAuth 2.0 Authorization Server Issuer Identification
- [RFC9700] Best Current Practice for OAuth 2.0 Security



FAQ 2: Isn't it mix-up attack?
Paradigm Shift: Reflections on Trust

Traditional OAuth for Single Sign-on (SSO)



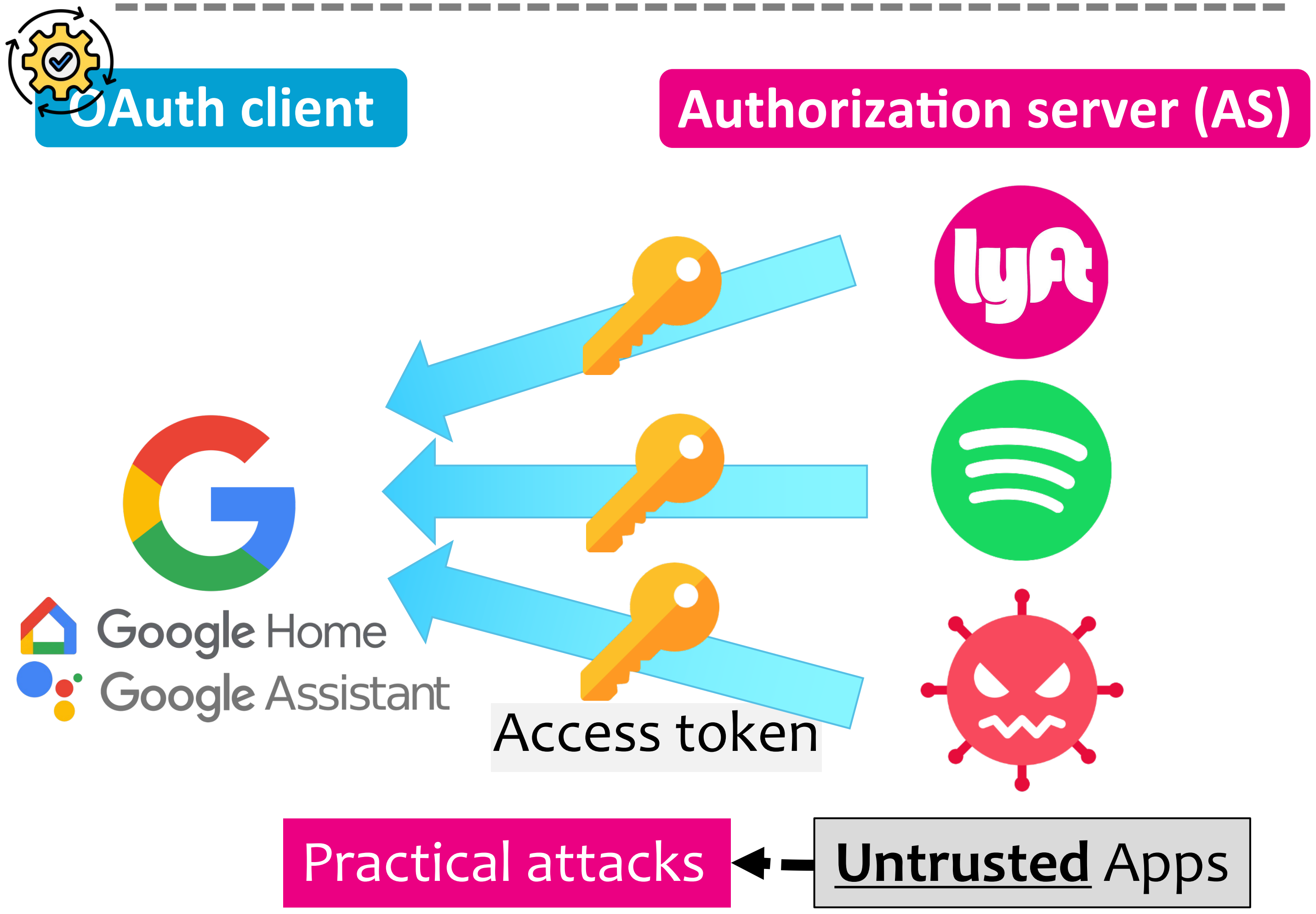
OAuth client
a.k.a. Relying party (RP)

Authorization server (AS)
a.k.a. Identity provider (IdP)

Access token

Theoretical attacks ← **Trusted** IdPs

⭐ OAuth for "Account Linking" in Integration Platforms



OAuth client

Authorization server (AS)

Access token

Practical attacks ← **Untrusted** Apps

- Multiple Auth Servers: Easy
- One of them is malicious: **Hard**

❖ (IdP) mix-up attack → Cross-app OAuth Account Takeover (COAT)

FAQ 2: Isn't it mix-up attack?

Suggested Changes: Attack Scenario

Extend the mix-up attack scenarios:

- The attacker uses **dynamic registration** to register the client at their own authorization server;

  Hard → e.g. Spotify adds malicious.com for sign in

- **[NEW]** The attacker exploits **open ecosystems** to register their own authorization server at the client for app integrations;

  Easy → e.g. Malicious app integrated with Google Assistant

- An authorization server becomes **compromised**.

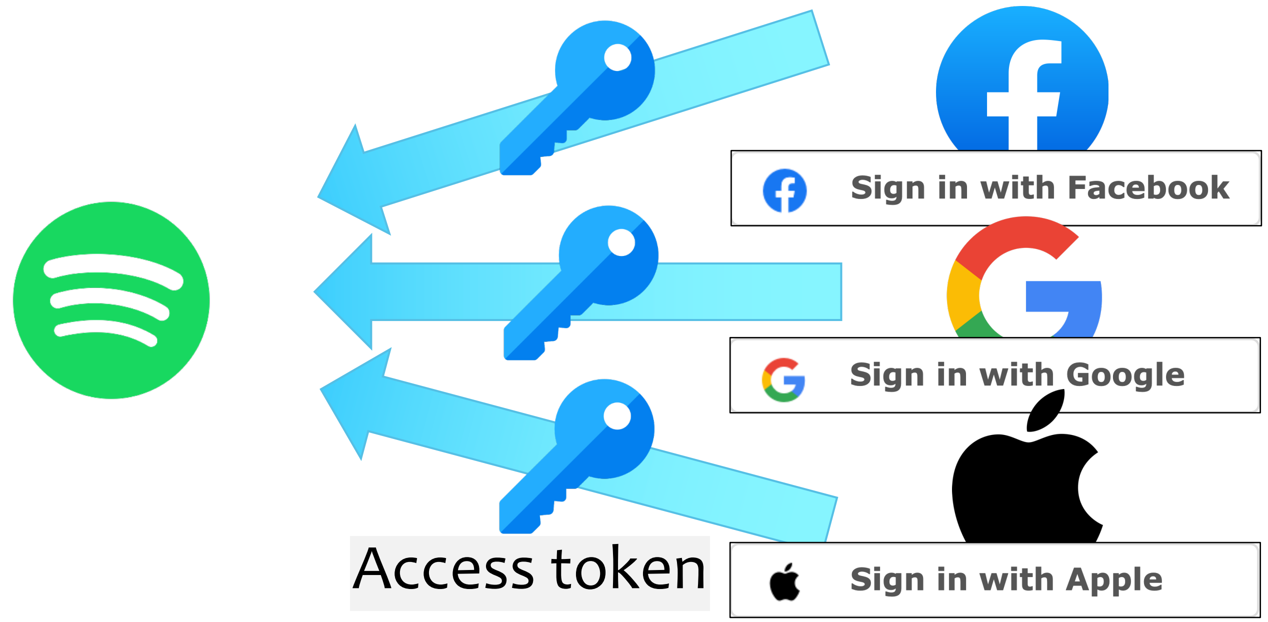
  Hard → e.g. "Sign in with Google" gets hacked

FAQ 2: Isn't it mix-up attack?
Paradigm Shift: Reflections on Trust

Traditional OAuth for Single Sign-on (SSO)

OAuth client
a.k.a. Relying party (RP)

Authorization server (AS)
a.k.a. Identity provider (IdP)



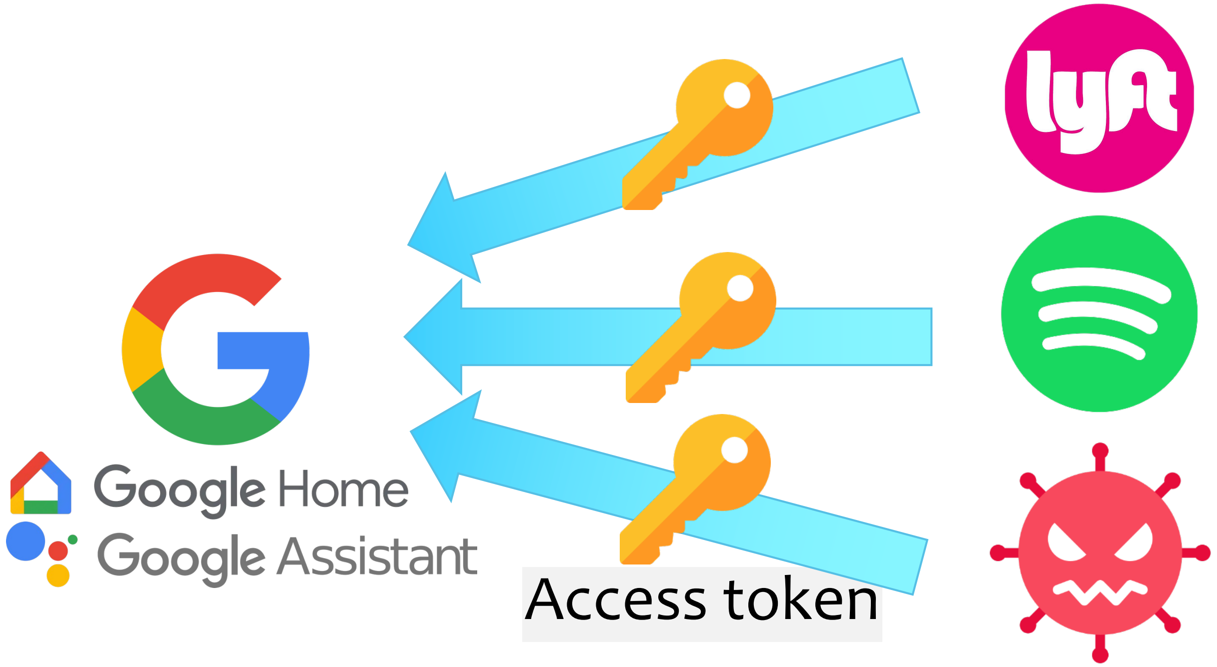
Theoretical attacks ← **Trusted** IdPs

- Multiple Auth Servers: Easy
- One of them is malicious: **Hard**

⭐ OAuth for "Account Linking" in Integration Platforms

OAuth client

Authorization server (AS)



Practical attacks ← **Untrusted** Apps

❖ (IdP) mix-up attack → Cross-app OAuth Account Takeover (COAT)
❖ Naïve RP session integrity attack → Cross-app OAuth Request Forgery (CORF)

FAQ 3: Why can't we use existing mix-up defense?

Review on Existing Countermeasures

Mix-Up Defense via Issuer Identification (Detailed in RFC9207)

vs. in integration platform: per-App ID, not per-AS ID

- Each Authorization Server returns a *unique ID in* authorization response (the *issuer* identifier)
- Client knows the expected *issuer*, and ensures this ground truth is **trustworthy** (e.g., sourced from OAuth Authorization Server metadata RFC8414)
- Client compares the returned *issuer* with the ground truth at the redirection endpoint

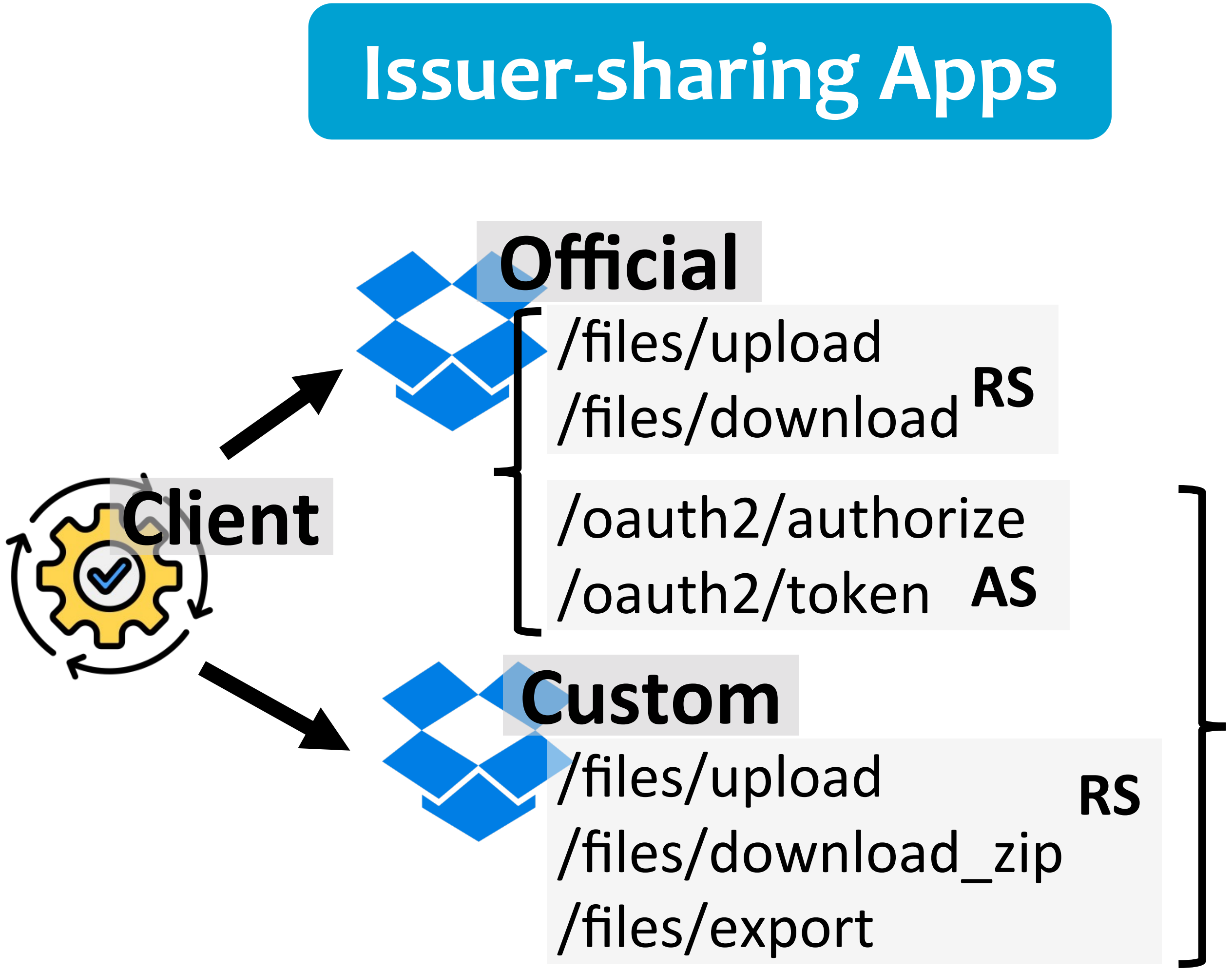
Mix-Up Defense via Distinct Redirect URIs

Basis of defense for Cross-app OAuth Attacks

- Client issues a distinct `redirect_uri` for each Authorization Server during OAuth registration, which serves as the ground truth
- Client compares the request URL (corresponding to `redirect_uri`) with the ground truth at the redirection endpoint

FAQ 3: Why can't we use existing mix-up defense?

Compatibility/scalability and responsibility concerns



FAQ 3: Why can't we use existing mix-up defense?
Compatibility/scalability and responsibility concerns

Mix-Up Defense via Issuer Identification (Detailed in RFC9207)

vs. in integration platform: per-App ID, not per-AS ID

- Each Authorization Server returns a *unique ID in* authorization response (the *issuer* identifier)
- Client knows the expected *issuer*, and ensures this ground truth is **trustworthy** (e.g., sourced from OAuth Authorization Server metadata RFC8414)
- Client compares the returned *issuer* with the ground truth at the redirection endpoint

vs. in integration platform: manual registration

Why is the *Issuer Identification* defense not practical?

- **Two apps can share *issuer*:** *issuer* (per-AS ID) is not unique; per-App ID is unique.
- **Scalability and responsibility concerns:** No trusted ground truth for *issuers*, as most apps lack (latest) standards-compliance. Better shift responsibilities from apps to platform.

FAQ 3: Why can't we use existing mix-up defense?

Suggested Changes: Countermeasure

The user's choice { **stored by the Client** / **returned by the Authorization Server** } **could be not only an *authorization server,* but also *an app*.**

Functional Requirement

Security Requirement

Practical Defense Based on *Mix-Up Defense via Distinct Redirect URIs*

- Use a ***per-app ID*** rather than a ***per-authorization server ID (issuer)***, to better reflect the multi-app nature of integration platforms.

- To maximize compatibility, impose ***no new dependencies*** on apps' authorization servers already compliant with the original OAuth spec [RFC6749].

⇒ Essential for securing platforms integrated with hundreds of apps,
potentially with shared *issuers.*

- Defense ***also applies to CORF***/Naïve RP session integrity attack.

Key Takeaways

- As ***open ecosystems***, Cross-app OAuth Attacks in Integration Platforms enable practical variants of Mix-up Attacks via ***malicious app integrations***.
- Existing RFCs have ***AS-centric*** defense but lack ***app-centric*** defense.
  A per-app ID is the correct isolation boundary for multi-app integrations.
- With 15+ vulnerable mainstream platforms identified and Hundreds/thousands of integrated apps per platform:
  - Pervasive Impact across the Internet;
  - Better rely on the ***platform*** *(client)* rather than ***individual apps*** *(AS)* for the defense.

⇒ **Next Steps**: Revision to the OAuth Security BCP?

More Info

USENIX Security '25 paper: Full-blown Analysis Vulnerability Detection

"Universal Cross-app Attacks: Exploiting and Securing OAuth 2.0 in Integration Platforms."

Kaixuan Luo, Xianbo Wang, Pui Ho Adonis Fung, Wing Cheong Lau, and Julien Lecomte.
To appear in 34th USENIX Security Symposium, August 2025.

Black Hat USA '24 talk: Attack-centric Style Other Interesting Issues

"One Hack to Rule Them All: Pervasive Account Takeovers in Integration Platforms for Workflow Automation, Virtual Voice Assistant, IoT, & LLM Services."

Video: https://www.youtube.com/watch?v=qrHEBElig3c

Slides: https://i.blackhat.com/BH-US-24/Presentations/US24-Luo-One-Hack-to-Rule-Them-All-Thursday.pdf

Cross-app OAuth Attacks in Integration Platforms:

Mix-up Attacks Reloaded

- *Research paper*
- *Full texts of proposed spec changes to IETF OAuth Security BCP*
- *This slide deck*

https://mobitec.ie.cuhk.edu.hk/osw2025

Questions?

Kaixuan Luo[1] **kaixuan@ie.cuhk.edu.hk**

Xianbo Wang[1], Adonis Fung[2], Julien Lecomte[2], Wing Cheong Lau[1]

1 The Chinese University of Hong Kong, 2 Samsung Research America

More on "Mix-up Attacks Reloaded" (Unconference Session)

Suggested Spec Changes based on Security BCP*

Section 4.4. Mix-Up Attacks

#1 – Attack Scenario (Lead Paragraph of Section 4.4.)

#2 – Attack Description (Section 4.4.1.)

#3 – Countermeasure (Section 4.4.2.)

#4 – Others (Section 4.4.1.)

* Updates based on the published version of OAuth Security BCP RFC9700 : https://datatracker.ietf.org/doc/html/rfc9700

Suggested Spec Changes #1 – Attack Scenario

Section 4.4. Mix-Up Attacks

New texts in red

Changes:

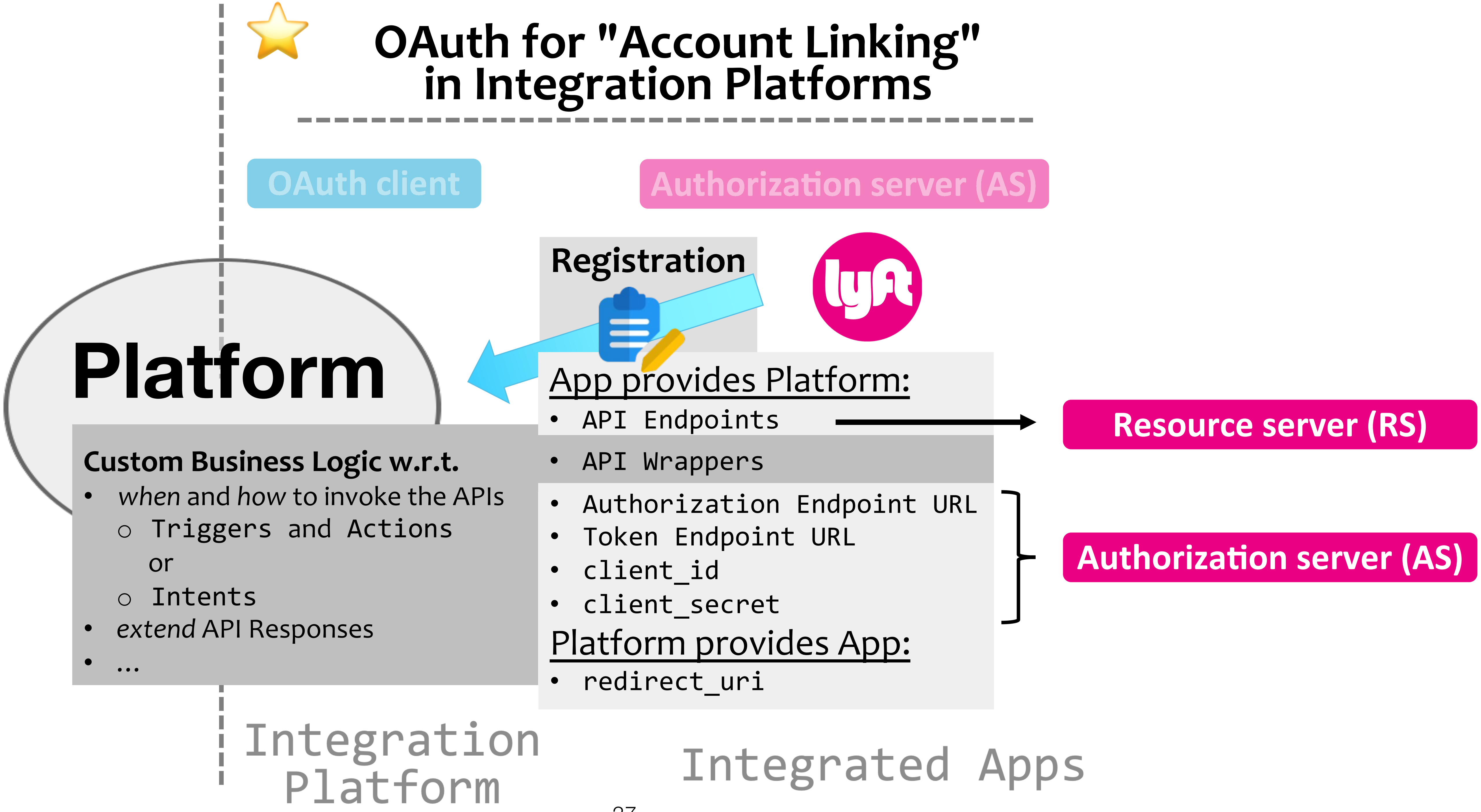
This can be the case, for example, if the attacker uses dynamic registration to register the client at their own authorization server**, if the attacker exploits open ecosystems to register their own authorization server at the client for app integrations,** or if an authorization server becomes compromised.

Rationale:

Extend the mix-up attack scenarios, to reflect the possibility of proactively introducing attacker-controlled authorization servers in open ecosystems like integration platforms.

Suggested Spec Changes #2 – Attack Description

App Integration: More than <AS, RS>



OAuth for "Account Linking" in Integration Platforms

Custom Business Logic w.r.t.

- *when* and *how* to invoke the APIs
  - ○ `Triggers and Actions`
    or
  - ○ `Intents`
- *extend* API Responses
- …

App provides Platform:

- `API Endpoints`
- `API Wrappers`
- `Authorization Endpoint URL`
- `Token Endpoint URL`
- `client_id`
- `client_secret`

Platform provides App:

- `redirect_uri`

Suggested Spec Changes #2 – Attack Description

Section 4.4.1. Attack Description

Rationale:

Functional Requirement: The user's choice **stored by the client** could be not only an *authorization server,* but also *an app*.

- For *attack precondition*, clarified that in multi-app integration ecosystems, app is the correct isolation boundary: they may share the same authorization server, but handle the requests to resource server differently. e.g.,

  ① Sending to *different resource servers* such as an API gateway first,

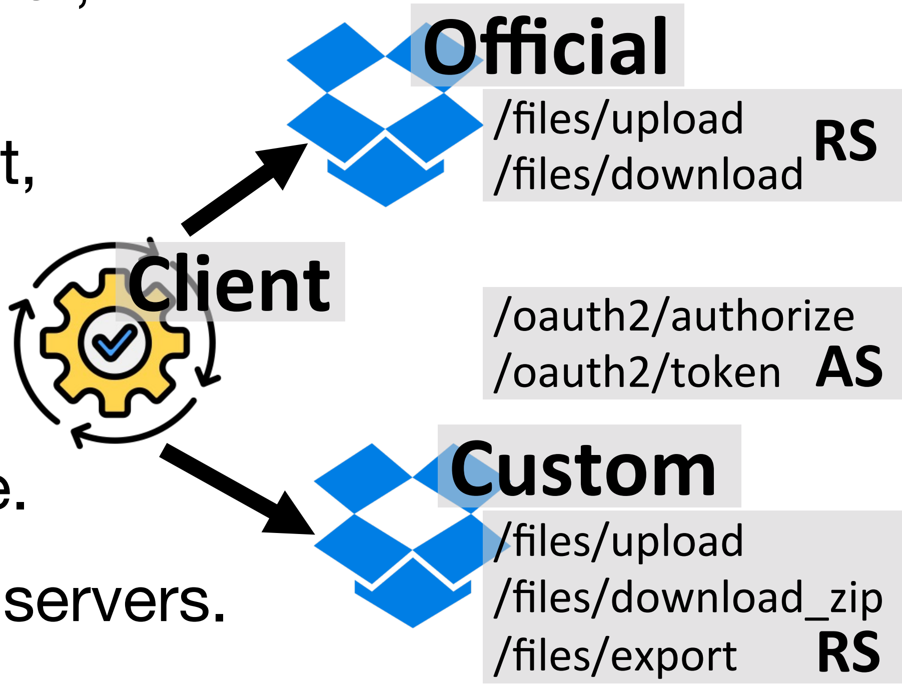
  ② Sending to *different API endpoints of the same resource server*,

  ③ Sending to the APIs with *different request parameters*,

  ④ Having *different wrappers around the same APIs* at the client side.

⇒ Motivates the client to ***differentiate by apps*** rather than authorization servers.

- For *attack description*, added a brief attack scenario description and pointed to our USENIX Security '25 paper for further reference.



Suggested Spec Changes #2 – Attack Description

Section 4.4.1. Attack Description

New texts in red

Changes:

Variants:

- Mix-Up with Interception: …
- Per-AS Redirect URIs: …
- Implicit Grant: …
- OpenID Connect: …
- **Multi-app Integration Ecosystem:** In ecosystems such as workflow automation platforms or virtual assistants, a client integrates with multiple pairs of authorization and resource servers that function as connected apps. While several apps may share the same authorization server, each app requires the client to interact with the corresponding resource server in different ways. **To handle each app independently, the client needs to treat shared authorization servers as separate servers and obtain authorization codes or access tokens from each individually.**

  In these scenarios, the client typically stores the **selected app** instead of the **selected authorization server** in the user's session. Attackers can mount a mix-up attack by targeting the H-AS of an uncompromised app with the A-AS of a malicious or compromised app. For details on this attack vector, see Section 4.2.1 of [research.cuhk] ("Cross-app OAuth Account Takeover").

Suggested Spec Changes #3 – Countermeasure

Section 4.4.2. Countermeasures

Section 4.4.2.1. Mix-Up Defense via Issuer Identification

- Each Authorization Server returns a unique ID in the authorization response (the *issuer* identifier)

vs. in integration platform: per-App ID, not per-AS ID

- Client knows the expected issuer when initiating OAuth, and this ground truth is trustworthy (e.g., sourced from OAuth Authorization Server metadata RFC8414)

vs. in integration platform: Manual Registration!

- Client compares the returned *issuer* with the ground truth at the redirection endpoint
- Detailed in *OAuth 2.0 Authorization Server Issuer Identification* RFC9207

Section 4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

Basis of defense for *Cross-app OAuth Attacks*

- Client issues a distinct redirect_uri for each Authorization Server during OAuth registration, which serves as the ground truth
- Client compares the request URL (corresponding to the redirect_uri) with the ground truth at the redirection endpoint

Suggested Spec Changes #3 – Countermeasure

Section 4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

New texts in red

Changes:

For this defense, clients MUST use a **distinct redirection URI for each issuer** they interact with.

Clients MUST check that the authorization response was received from the **correct issuer** by comparing the **distinct redirection URI for the issuer** to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

…

Note that for the **mix-up variant** in multi-app integration ecosystem (see Section 4.4.1), where an issuer is not always unique to a client, **a variant of this defense** is RECOMMENDED:

Clients SHOULD use a **distinct redirection URI for each app** they interact with,

and SHOULD check that the authorization response was received from the **correct app** by comparing the **distinct redirection URI for the app** to the URI where the authorization response was received on. If there is a mismatch, the client MUST abort the flow.

Suggested Spec Changes #3 – Countermeasure

Section 4.4.2.2. Mix-Up Defense via Distinct Redirect URIs

Security Requirement: The user's choice **returned by the Authorization Server** could be not only an *authorization server,* but also *an app*.

Rationale:

- Specifies the use of a ***per-app identifier*** rather than ***per-authorization server (issuer)***, to better reflect the multi-app nature of integration platforms.
- To maximize compatibility, it imposes ***no new dependencies*** on apps' authorization servers that are already compliant with the original OAuth spec [RFC6749]. This is essential for securing platforms that are integrated with thousands of apps.
- Defense ***also applies to CORF***/Naïve RP session integrity attack.
- We currently mark this variant defense as RECOMMENDED/SHOULD, which is open for discussion.

Suggested Spec Changes #4 – Others

Common misunderstanding:

distinct redirect_uris -> no mix-up;
or, mix-up -> shared redirect_uri ✗

Ref: [OSW '16] Does the IdP Mix-Up attack really work?; [EuroS&P '24] SSO-Monitor;
[RFC9700] OAuth Security BCP

e.g., "*Preconditions: the client uses the same redirection URI for each authorization server.*"

We Clarify that:

(1) mix-up could still happen if distinct redirect_uri is used (COAT_D, or Slide #15)

(2) [TODO] Even if not susceptible to mix-up, the use of distinct redirect_uri could still result in CORF / Naïve RP Session Integrity Attack

Suggested Spec Changes #4 – Others

Section 4.4.1. Attack Description

New texts in red

Changes:

Variants:

- …
- Per-AS Redirect URIs: When clients use **different redirection URIs for different authorization servers but treat them as the same URI, the attack would still work.** An attacker can achieve this by replacing the redirection URI as well as the client ID at A-AS with those at H-AS in the authorization request in Step 3.

  Alternatively, if clients use different redirection URIs for different authorization servers, clients do not store the selected authorization server in the user's session, and authorization servers do not check the redirection URIs properly, attackers can mount an attack called "Cross Social-Network Request Forgery" (refer to [research.jcs_14] for details). These attacks have been observed in practice.