

OAuth Checklist



OAuth Checklist

Table Of Contents

- OAuth Roles
- Code Flaws
- Redirect_uri Flaws
- State Flaws
- Evil App
- Misc

By: HolyBugx

▼ OAuth

▼ OAuth Roles

- Resource Owner → User
- Resource Server → Twitter
- Client Application → Twitterdeck.com
- Authorization Server → Twitter
- client_id → Twitterdeck ID (This is a public, non-secret unique identifier.)
- client_secret → Secret Token known to the Twitter and Twitterdeck to generate `access_tokens`.
- response_type → Defines the token type e.g (code, token, etc.)
- scope → The requested level of access Twitterdeck wants.
- redirect_uri → The URL user is redirected to after the authorization is complete.
- state → Main CSRF protection in OAuth, can persist data between the user being directed to the authorization server and back again.
- grant_type → Defines the `grant_type` and the returned token type.
- code → The authorization code twitter generated, will be like `?code=`, the code is used with client_id and client_secret to fetch an `access_token`.
- access_token → The token twitterdeck uses to make API requests on behalf of the user.
- refresh_token → Allows an application to obtain a new `access_token` without prompting the user.

▼ Code Flaws

- ☐ Re-Using the code.
- ☐ Code Predict/Bruteforce and Rate-limit?
- ☐ Is the code for application X valid for application Y?

▼ Redirect_uri Flaws

- ☐ URL isn't validated at all: `?redirect_uri=https://attacker.com`
- ☐ Subdomains allowed (Subdomain Takeover or Open redirect on those subdomains): `?redirect_uri=https://sub.twitterdeck.com`
- ☐ Host is validated, path isn't (Chain open redirect): `?redirect_uri=https://twitterdeck.com/callback?redirectUrl=https://evil.com`
- ☐ Host is validated, path isn't (Referer leakages): Include external content on HTML page and leak code via `Referer`.
- ☐ Weak Regexes:

```
?redirect_uri=https://twitterdeck.com.evil.com
?redirect_uri=https://twitterdeck.com%252eevil.com
?redirect_uri=https://twitterdeck.com//evil.com/
?redirect_uri=https://twitterdeck.com%09evil.com
```

- ☐ Bruteforcing the URL encoded chars after host: `?redirect_uri=https://twitterdeck.com$FUZZ$`
- ☐ Bruteforcing the keywords whitelist after host (or on any whitelist open redirect filter): `?redirect_uri=https://$FUZZ$.com`
 - Imagine "twitter" keyword is allowed so we use: `?redirect_uri=https://eviltwitter.com`
- ☐ URI validation in place: use typical open redirect payloads.

▼ State Flaws

- ☐ Missing `State` parameter? (CSRF)
- ☐ Predictable `State` parameter?
- ☐ Is `State` parameter being verified?

What is the CSRF workflow in case of `state` problems?

- Attacker generate a valid `authorization_code` link for himself and doesn't use it (doesn't forward the request)
- Attacker sends the link to the logged-in victim, and if the victim opens the link, attacker's OAuth account will be linked to victim's.

▼ Evil App

- ☐ Race condition when `code` is exchanged for `access_token`
- ☐ Race condition when `refresh_token` is exchanged for `access_token`
- ☐ If user revokes access, will code be also revoked?

▼ Misc

- ☐ Is `client_secret` validated?
- ☐ Are `client_secret`, `access_token`, `refresh_token` leaking somewhere?
- ☐ Pre ATO using facebook phone-number signup
 - Register on facebook using a phone-number and it settings add the victim's email address (do not verify it).
 - Use the facebook OAuth on the target website, it might be possible that the application doesn't verify the victim's email.
 - Reference
- ☐ No email validation Pre ATO
 - Register as the victim with his email and your desired password.

- The victim then tries to login using OAuth such as google or facebook.
- The application queries the database and respond with: `email already exists.` and links their account to the attackers.
- If there is no un-link option on the application, the attacker can always login on behalf of the user using OAuth even if they reset password.

▼ References

- [HackerScroll](#)
- [The wonderful world of OAuth](#)
- [Pentester.land write ups](#)

By: HolyBugx