
eritus
INFORMATION SECURITY SERVICES


bugcrowd
OUTHACK THEM ALL™

Hacking OAuth 2.0

For Fun And Profit

By Pranav Hivarekar

Peritus | bugcrowd level up conference

About Me

- **Pranav Hivarekar**
- **Active Bug Hunter**
  - Facebook (Top 10)
  - Snapchat, Shopify, Slack, and many more private programs...
- **MS degree in Cybersecurity from USA**
- **Worked at NCC Group, NYC, NY, USA**
- **Founder @PeritusInfosec**
- **Trainings @PeritusInfosec**
- **Runs @bugbounty_world**

AIM

A › B › C

Bug Hunters **Techniques** **Bounties**

Agenda

History of OAuth

OAuth 2.0 Basics

What is OAuth 2.0?
How OAuth 2.0 works?
Where OAuth 2.0 is used?

Attacks on OAuth 2.0 Integrations

Token/Code Stealing
CSRF (missing `state` param)
Token Impersonation

Conclusion

HISTORY OF OAuth

A

OpenID

(mainly designed as an authentication protocol)

B

OAuth

(authorization framework)

OAuth 1.0

OAuth 2.0

C

OpenID Connect

(built on top of OAuth 2.0)

OAuth 2.0 BASICS

What is OAuth 2.0?

How OAuth 2.0 works?

Where OAuth 2.0 is used?

What is OAuth 2.0?

Authorization Framework

It enables a third party application to obtain limited access to a service.

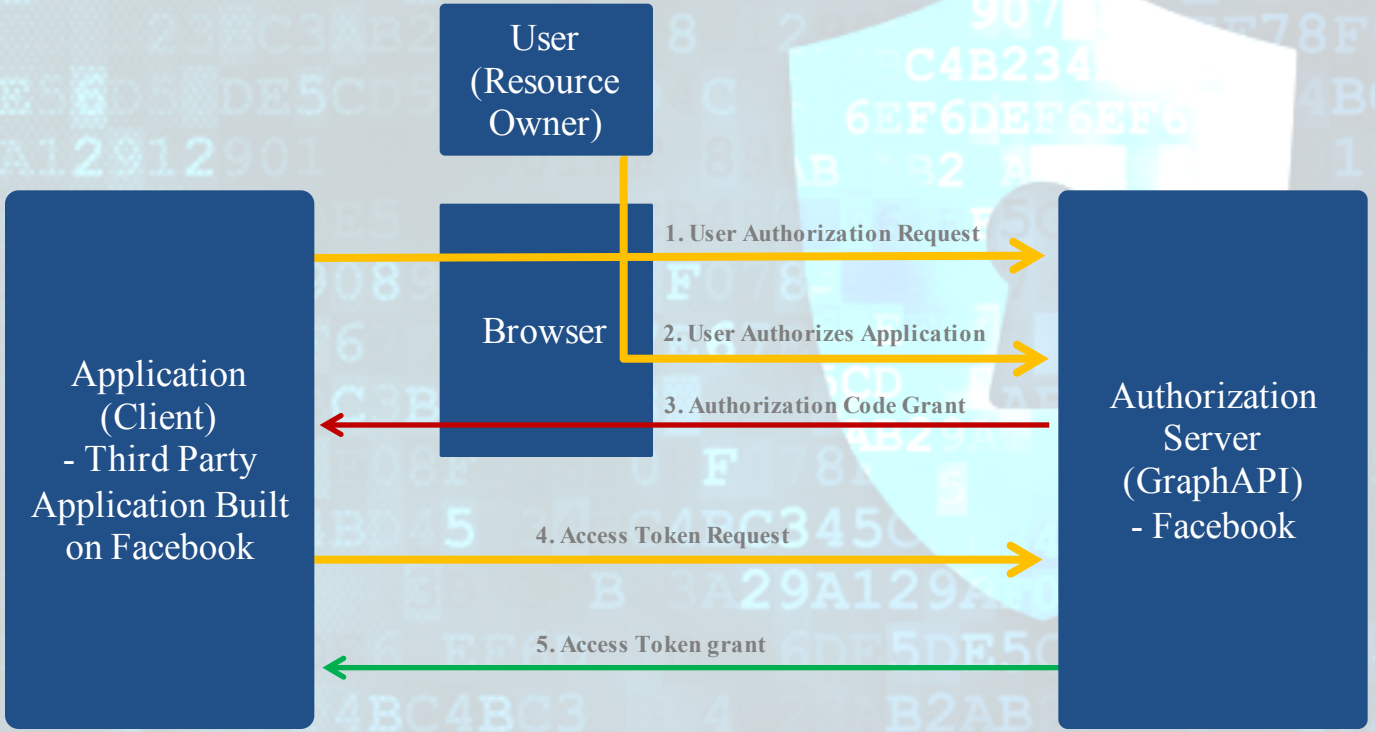
Example, we have `Login with Facebook` buttons on various websites which gets an `access_token` of the user from Facebook and uses limited information from Facebook to create an account.

Used by Facebook, Google, etc.

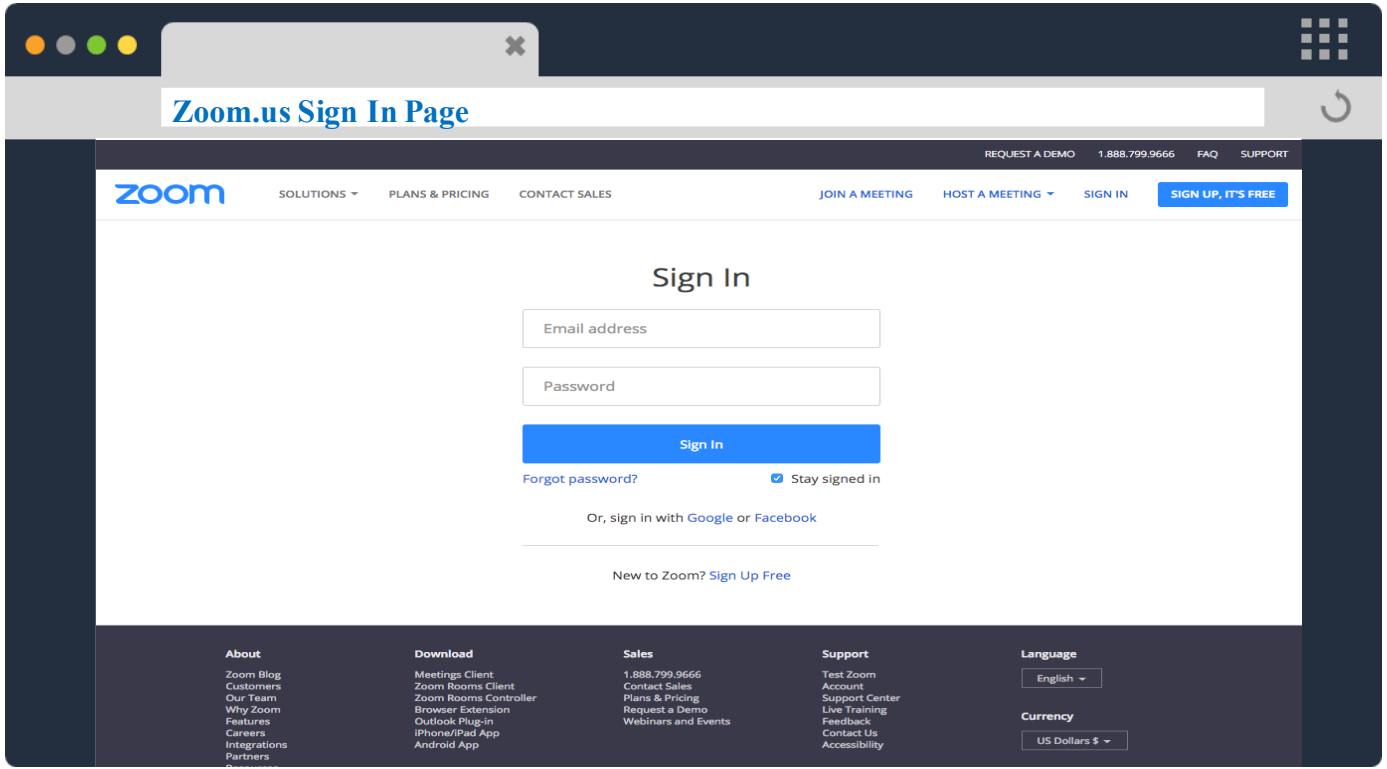
HOW OAuth 2.0 WORKS?

1. **Authorization Code**
2. **Implicit**
3. **Resource Owner Password Credentials**
4. **Client Credentials**

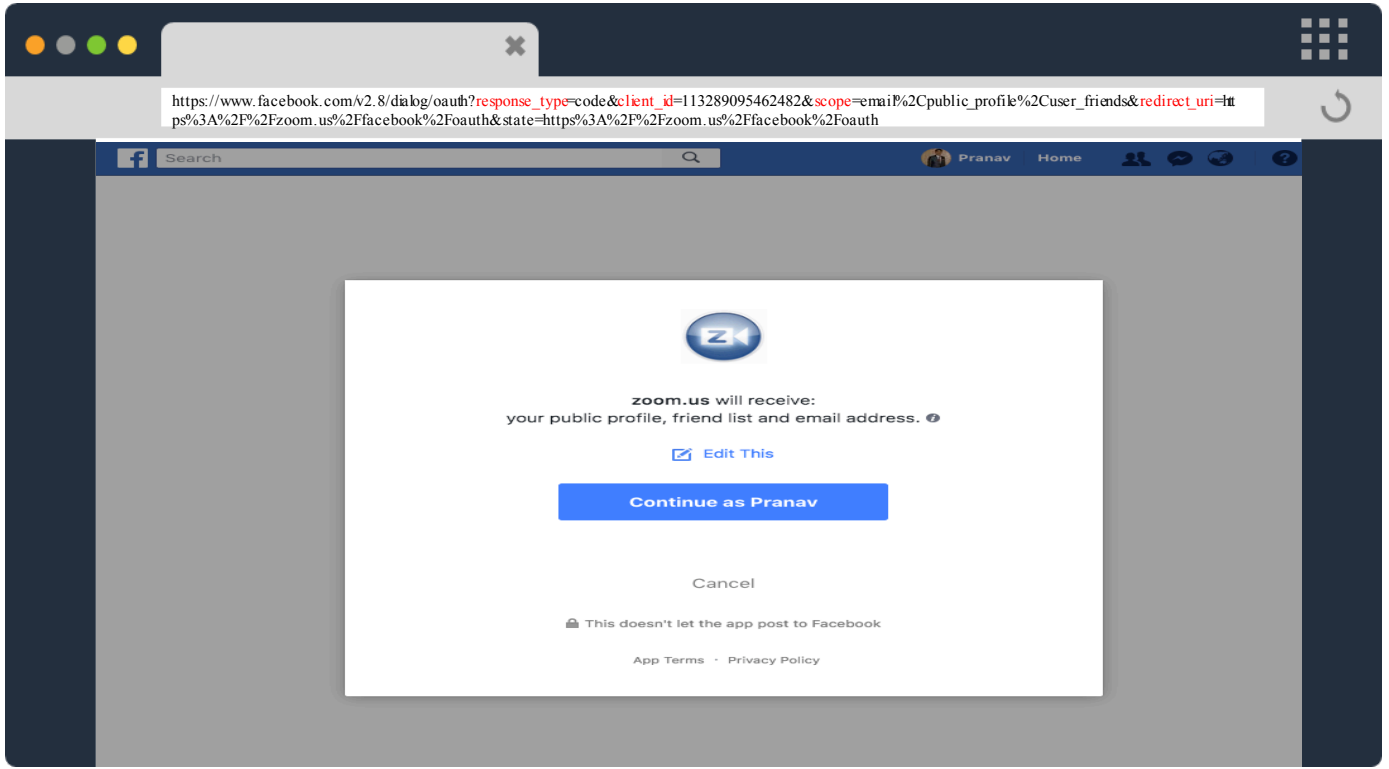
AUTHORIZATION CODE GRANT



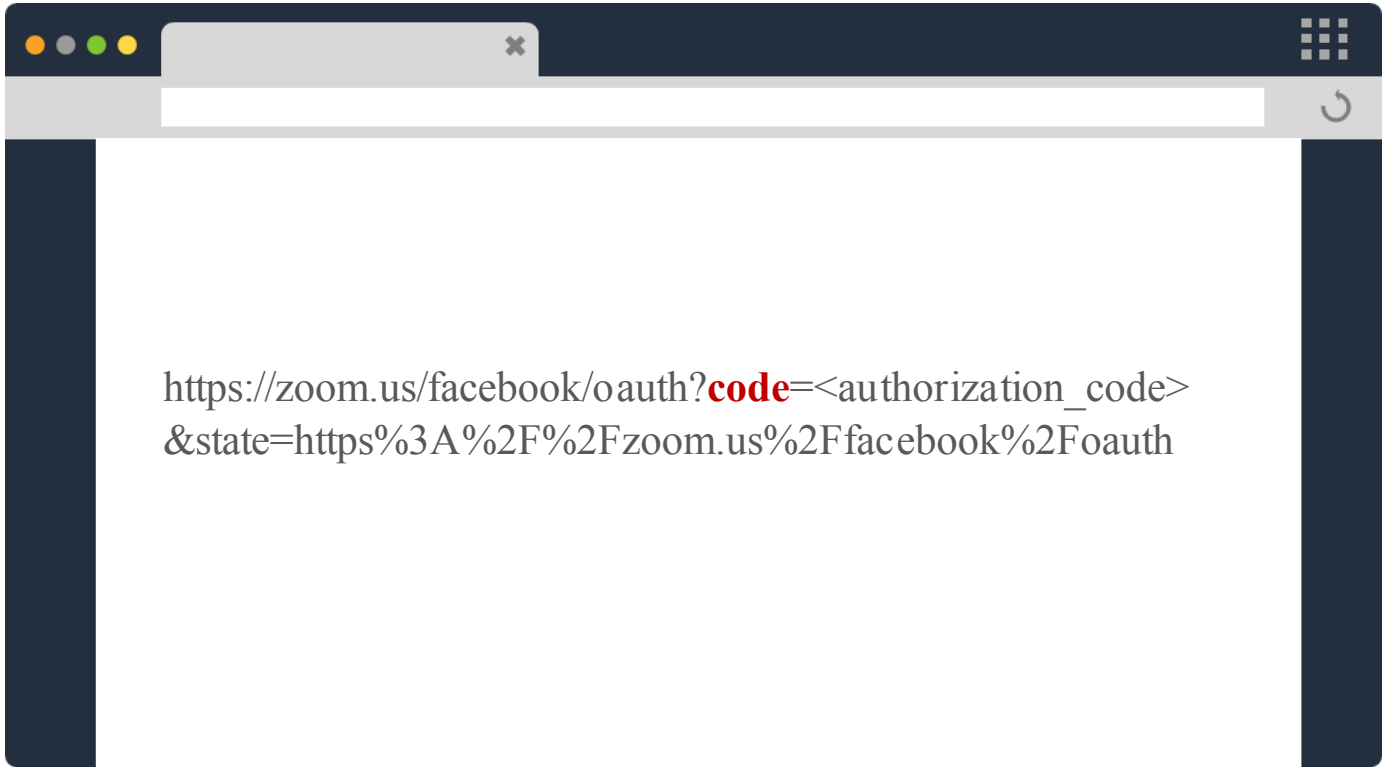
AUTHORIZATION CODE GRANT



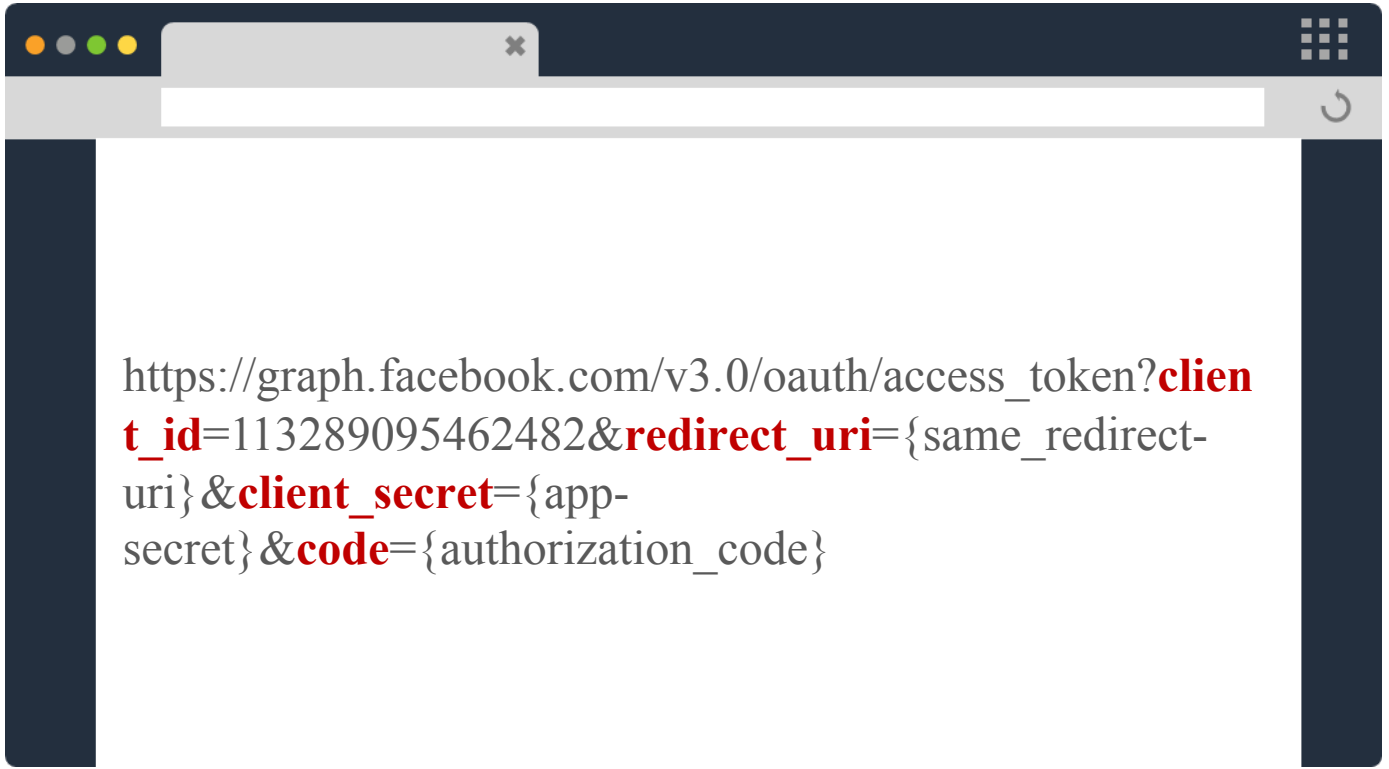
AUTHORIZATION CODE GRANT



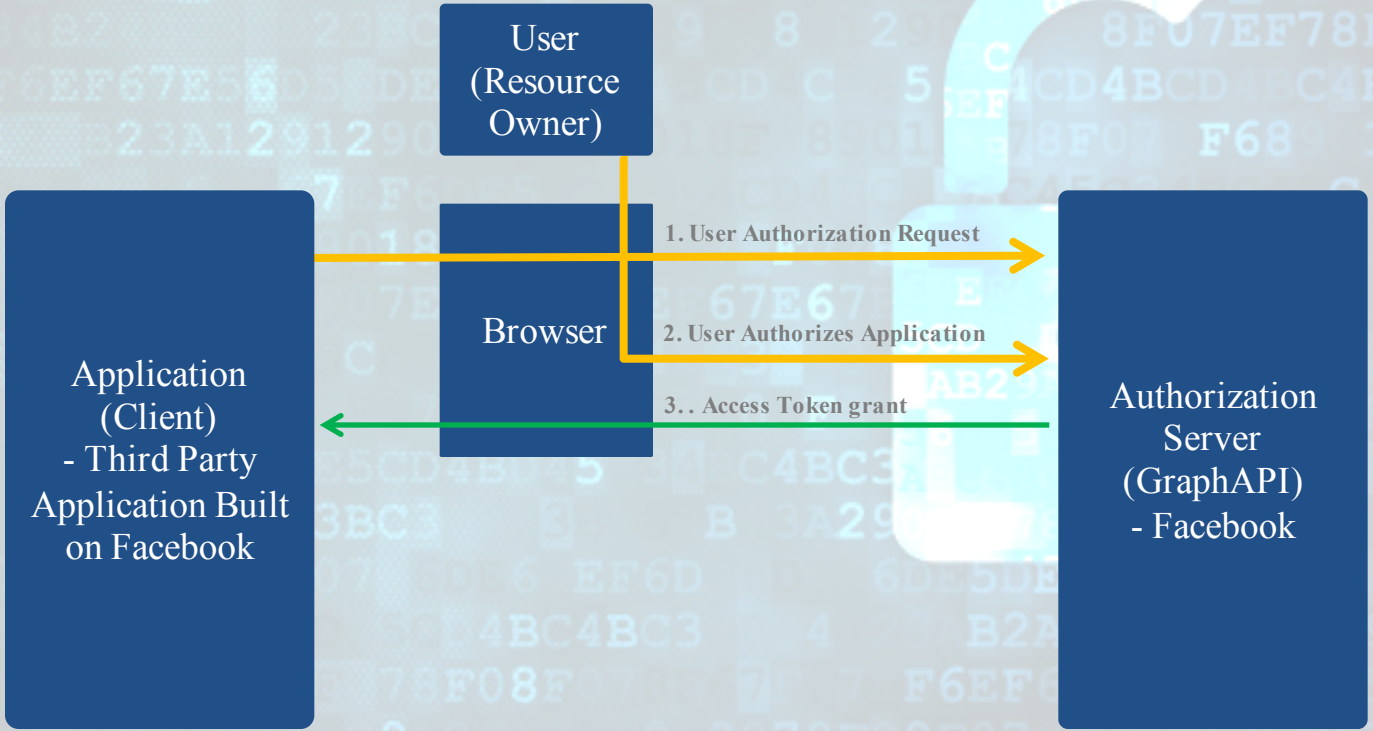
AUTHORIZATION CODE GRANT



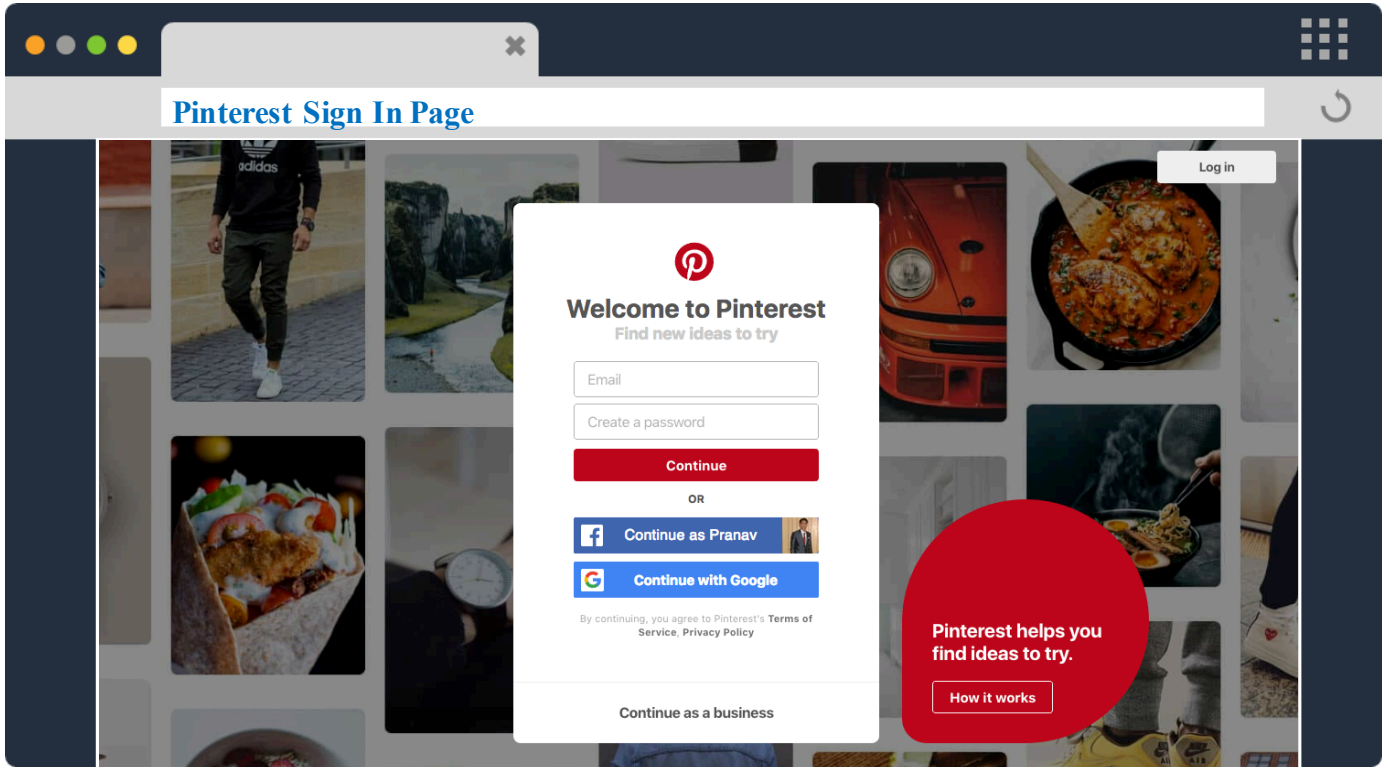
AUTHORIZATION CODE GRANT



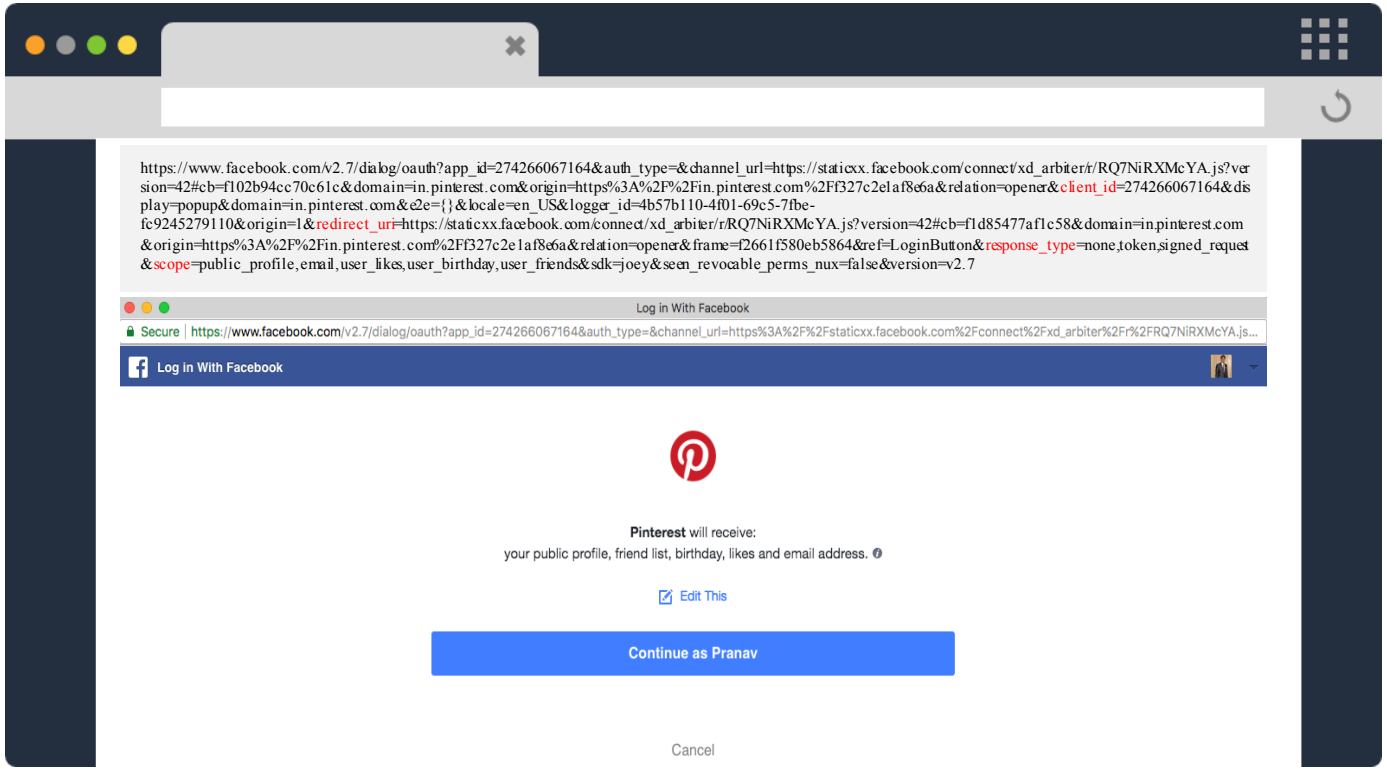
IMPLICIT GRANT



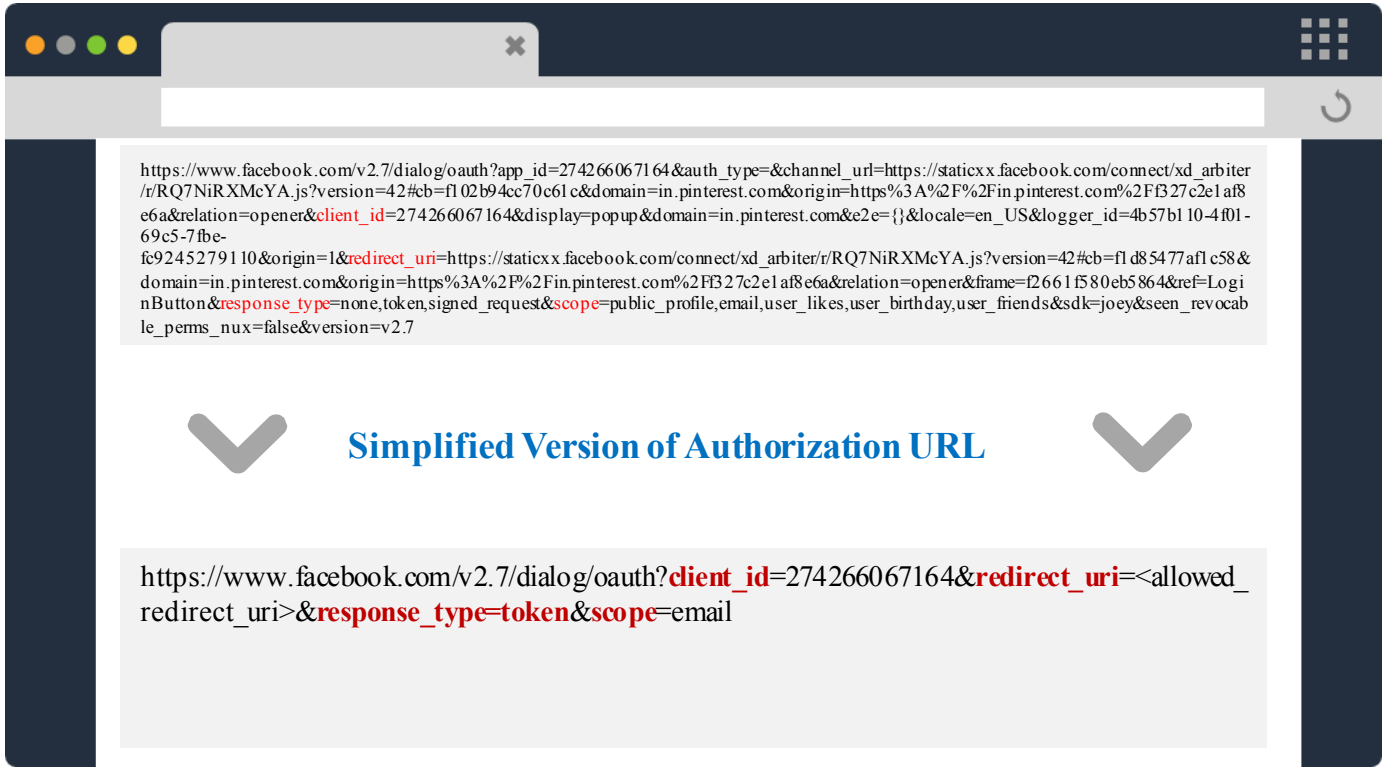
IMPLICIT GRANT



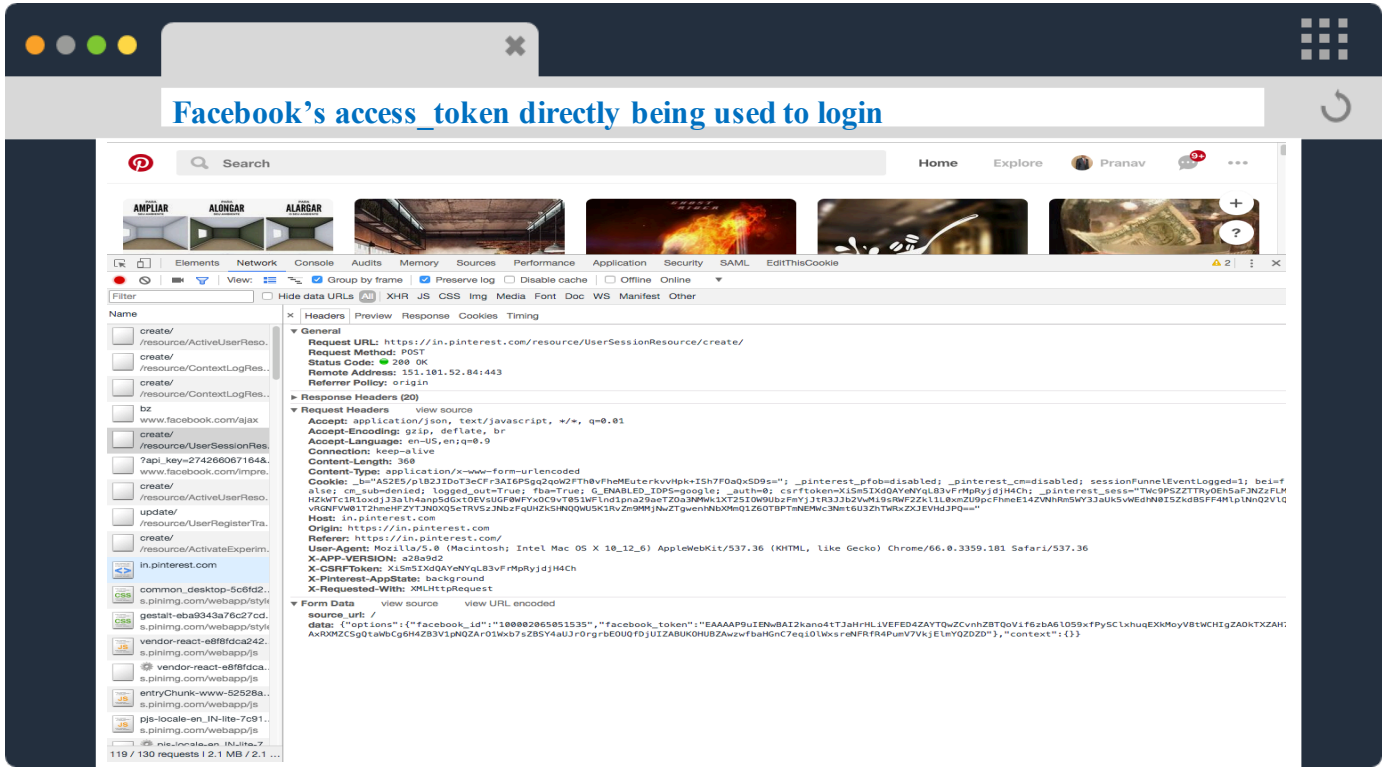
IMPLICIT GRANT



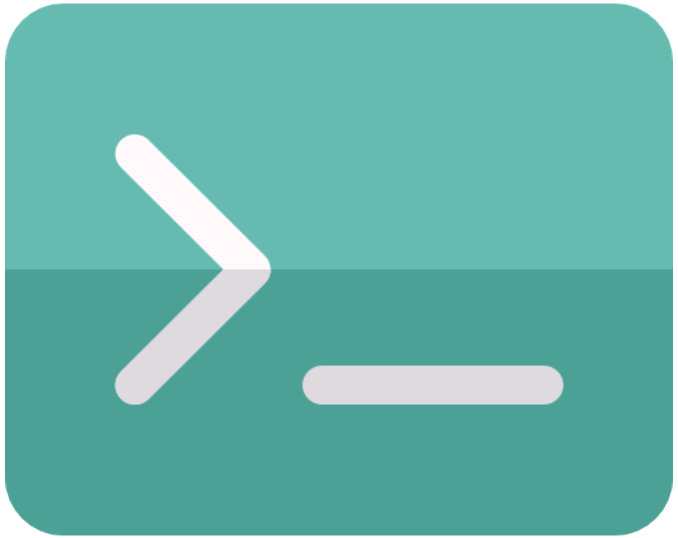
IMPLICIT GRANT



IMPLICIT GRANT



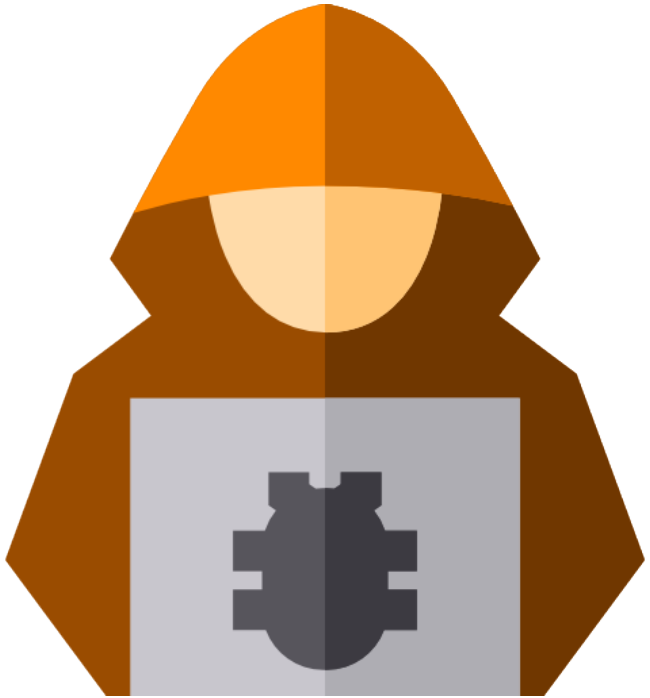
WHERE OAuth 2.0 IS USED?



1. Importing content from sites.
2. Login
3. SSO

ATTACKS ON OAuth 2.0 INTEGRATIONS

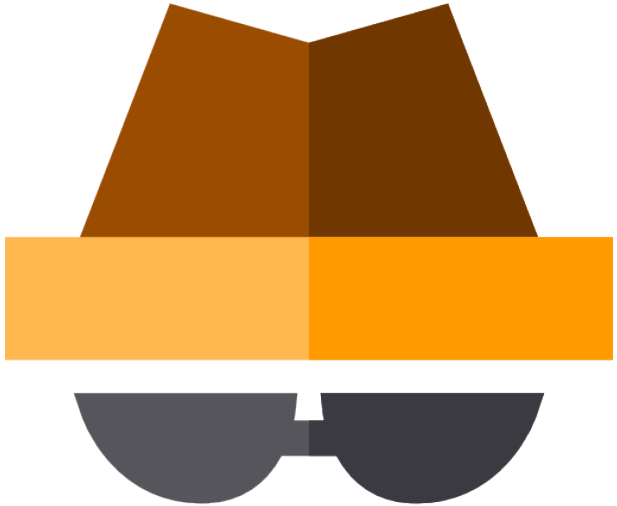
1. Token/Code Stealing
2. CSRF (missing `state` param)
3. Token Impersonation



TOKEN STEALING

1. What we do?
2. Secret Methodology
3. Case Study

TOKEN STEALING - What we do?

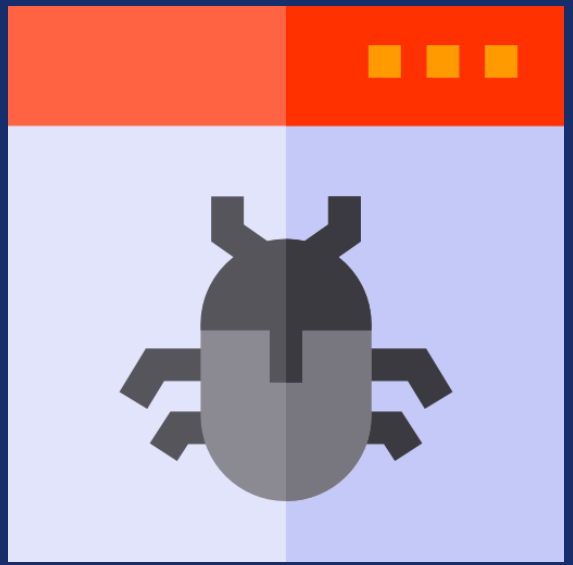


Our main AIM is to:

1. **Steal `access_token` of the application**
2. **Use it to login into user's account.**

https://www.facebook.com/v2.3/dialog/oauth?response_type=token&display=popup&client_id=<client_id>&redirect_uri=<redirect_uri>&scope=email

TOKEN STEALING - Secret Methodology



1. **Find domain used in `redirect_uri`.**
2. **Can you use `subdomains` in the `redirect_uri`.**
3. **Point the `redirect_uri` to a page.**
   a. Open redirector(302) to attacker's domain
   b. XSS which can be used in `redirect_uri` to pass `access_token` to attacker.
   c. Subdomain takeover (allowed subdomain in `redirect_uri`)
   d. Backtrack to a page which can be used to open redirect(302)/XSS
4. **Use the stolen `access_token` to login.**

Case Study

Bug Name : Stealing SSO Login Tokens (snappublisher.snapchat.com)

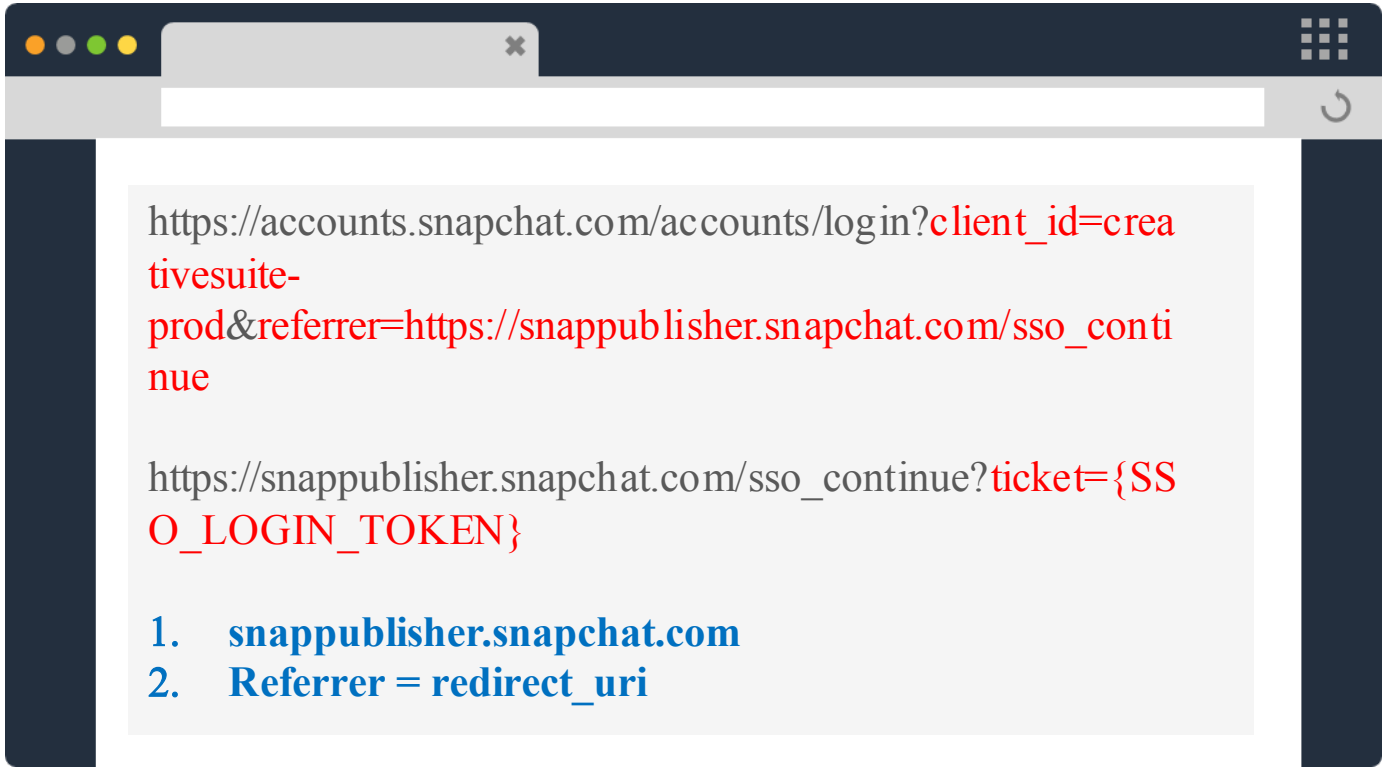
Program: Snapchat

Found by: Pranav Hivarekar(@HivarekarPranav)

Case Study

a. Background information
    a. Snapchat has different products like SnapPublisher, Ads Manager, Business Manager, etc.
    b. All these products fetched a `SSO LOGIN TOKEN` from main application(accounts.snapchat.com)
b. SSO Login functionality worked similar to that of OAuth 2.0.

PROOF OF CONCEPT

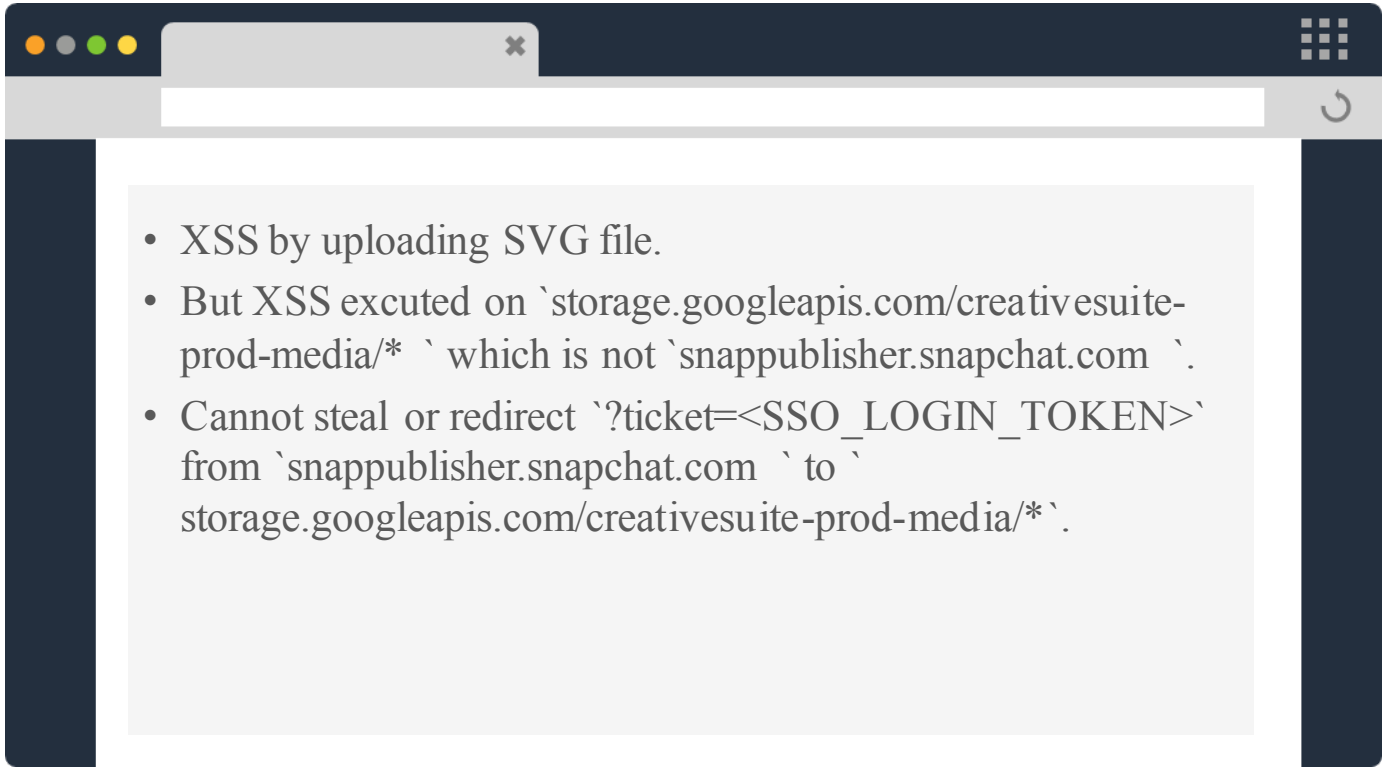


https://accounts.snapchat.com/accounts/login?client_id=creativesuite-prod&referrer=https://snappublisher.snapchat.com/sso_continue

https://snappublisher.snapchat.com/sso_continue?ticket={SSO_LOGIN_TOKEN}

1. **snappublisher.snapchat.com**
2. **Referrer = redirect_uri**

PROOF OF CONCEPT



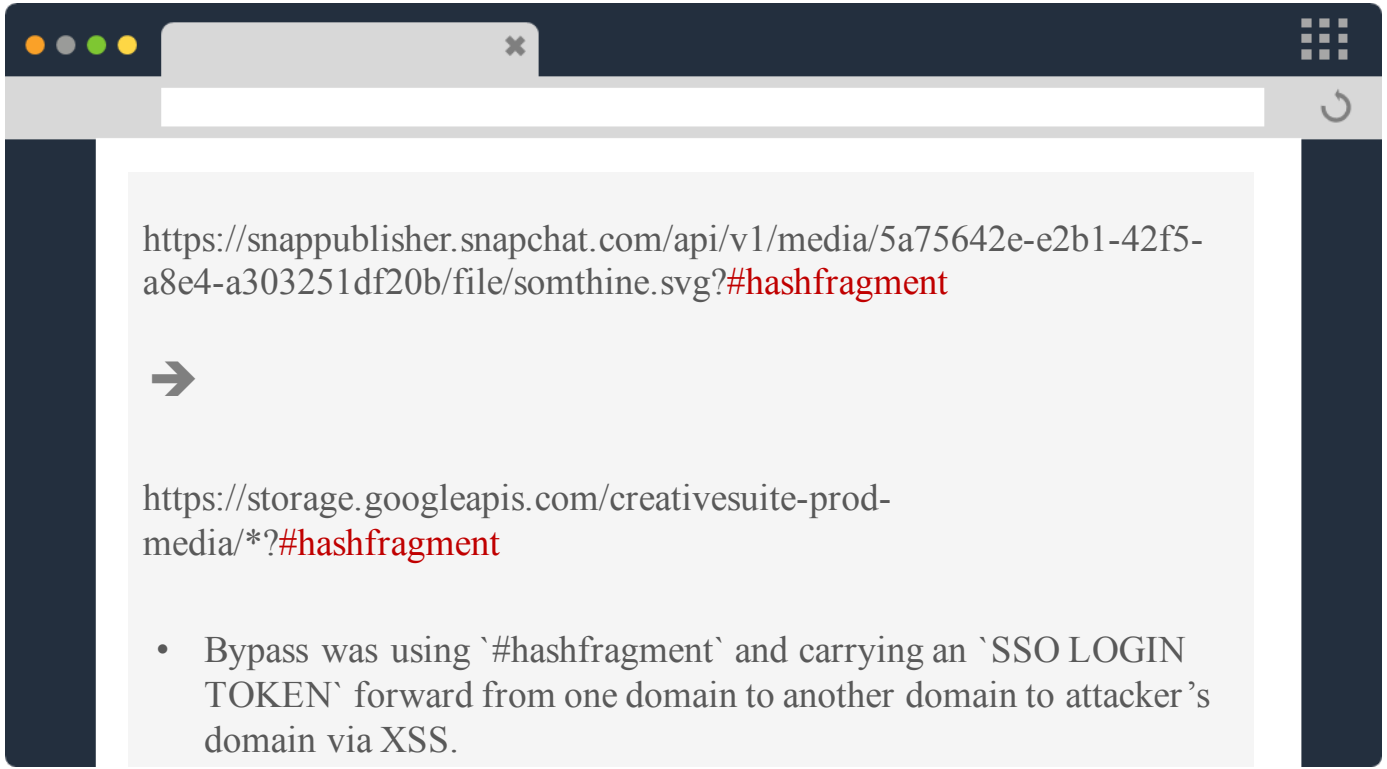
- XSS by uploading SVG file.
- But XSS excuted on `storage.googleapis.com/creativesuite-prod-media/* ` which is not `snappublisher.snapchat.com `.
- Cannot steal or redirect `?ticket=<SSO_LOGIN_TOKEN>` from `snappublisher.snapchat.com ` to ` storage.googleapis.com/creativesuite-prod-media/*`.

PROOF OF CONCEPT

My reaction

Source : https://www.askideas.com/wp-content/uploads/2016/11/Funny-Sad-Baby-Face-Photo.jpg

PROOF OF CONCEPT



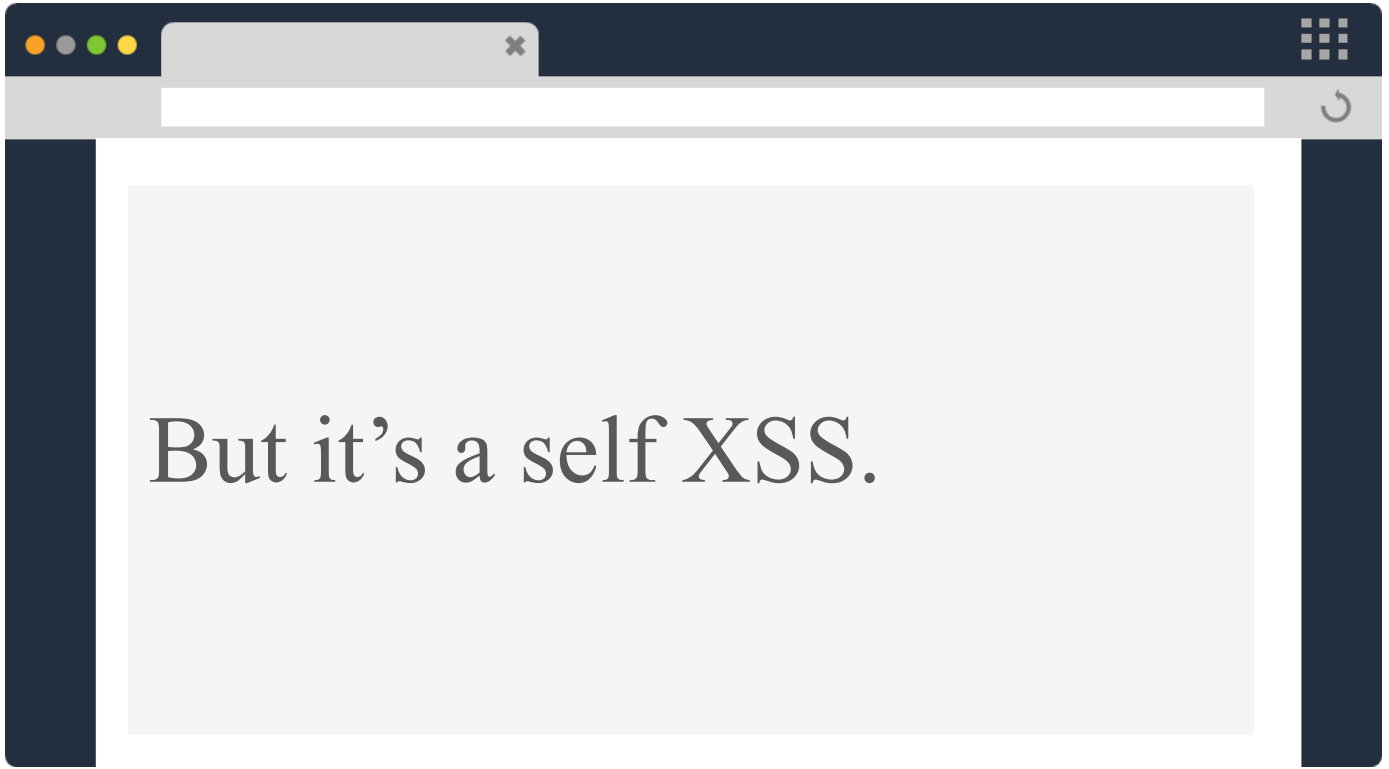
https://snappublisher.snapchat.com/api/v1/media/5a75642e-e2b1-42f5-a8e4-a303251df20b/file/somthine.svg?#hashfragment

➔

https://storage.googleapis.com/creativesuite-prod-media/*?#hashfragment

- Bypass was using `#hashfragment` and carrying an `SSO LOGIN TOKEN` forward from one domain to another domain to attacker's domain via XSS.

PROOF OF CONCEPT

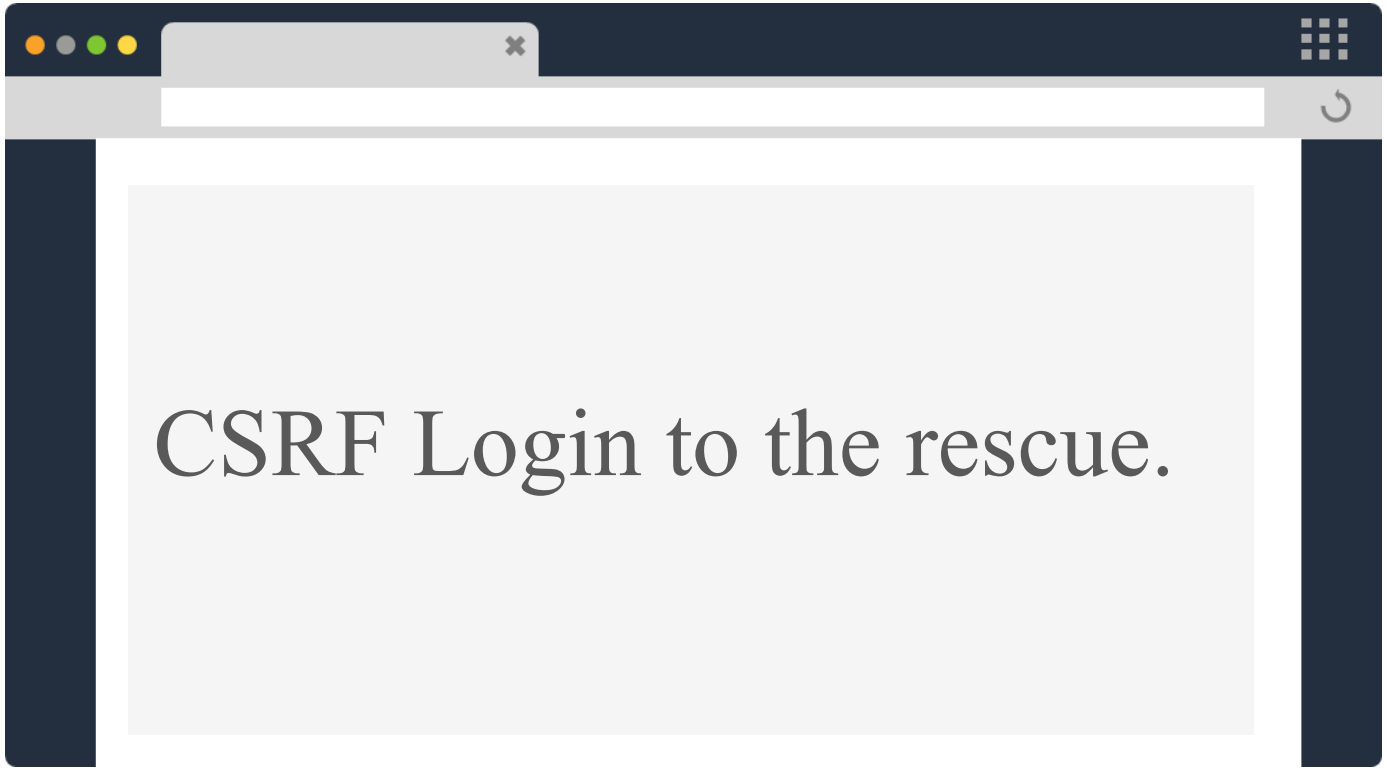


PROOF OF CONCEPT

My reaction

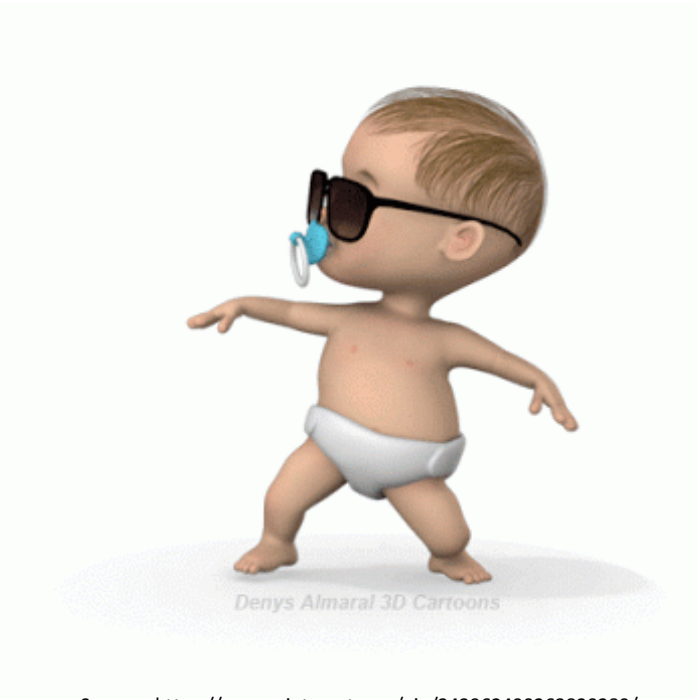
Source : http://compassionatesleepsolutions.com/wp-content/uploads/2013/04/crying-aby-with-hands-on-head-300x256.jpg

PROOF OF CONCEPT



PROOF OF CONCEPT

My reaction



Source : https://www.pinterest.com/pin/343962490263820289/

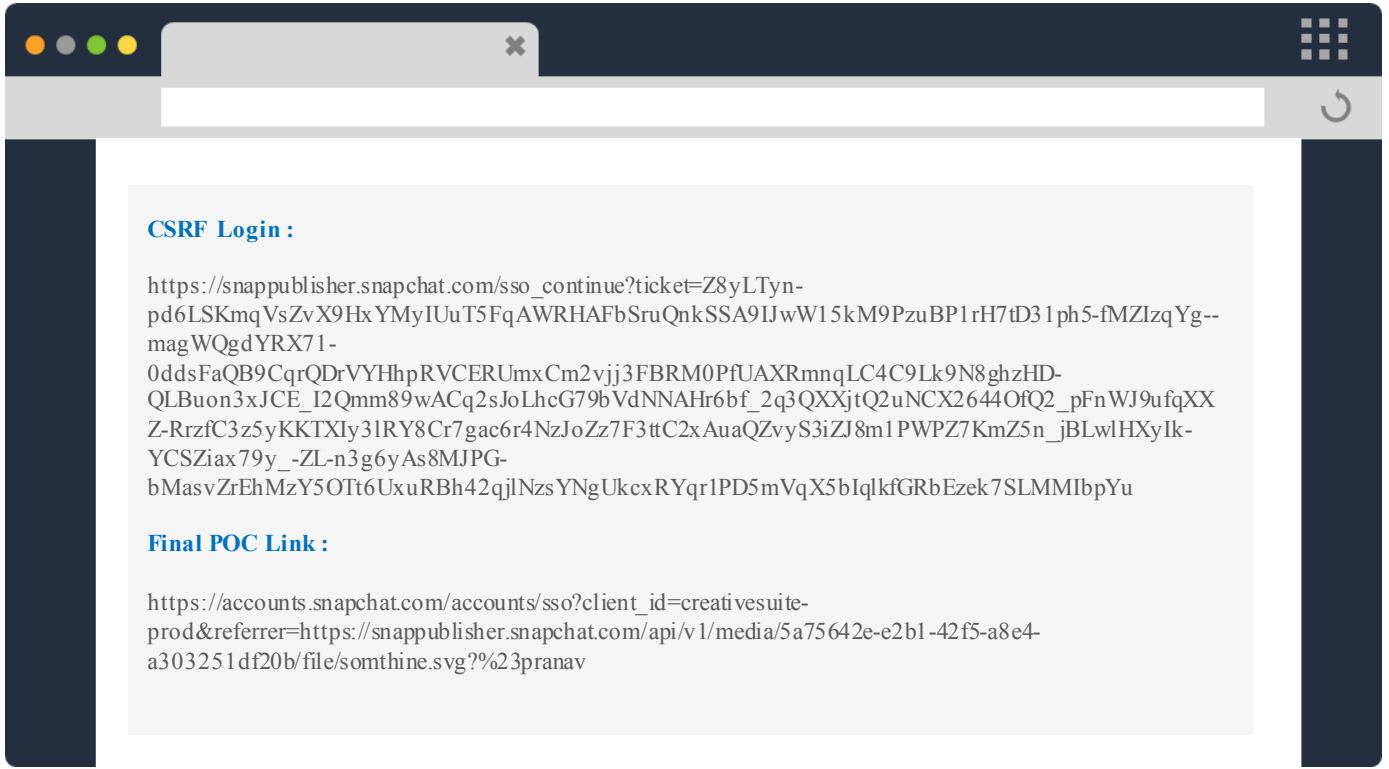
ATTACK WORKFLOW

Victim logins into main application (accounts.snapchat.com).

Attacker CSRF login's victim into his own account on snappubliser.snapchat.com.

Sends the token stealer link which steals victim's SSO LOGIN TOKEN received from main application by forwarding it from one domain to another and finally executing self-XSS

PROOF OF CONCEPT



CSRF Login :

https://snappublisher.snapchat.com/sso_continue?ticket=Z8yLTyn-pd6LSKmqVsZvX9HxYMyIUuT5FqAWRHAFbSruQnkSSA9IJwW15kM9PzuBP1rH7tD31ph5-fMZIzqYg--magWQgdYRX71-0ddsFaQB9CqrQDrVYHhpRVCERUmxCm2vjj3FBRM0PfUAXRmnqLC4C9Lk9N8ghzHD-QLBuon3xJCE_I2Qmm89wACq2sJoLhcG79bVdNNAHr6bf_2q3QXXjtQ2uNCX2644OfQ2_pFnWJ9ufqXXZ-RrzfC3z5yKKTXIy31RY8Cr7gac6r4NzJoZz7F3ttC2xAuaQZvyS3iZJ8m1PWPZ7KmZ5n_jBLwlHXyIk-YCSZiax79y_-ZL-n3g6yAs8MJPG-bMasvZrEhMzY5OTt6UxuRBh42qjlNzsYNgUkcxRYqr1PD5mVqX5bIqlkfGRbEzek7SLMMIbpYu

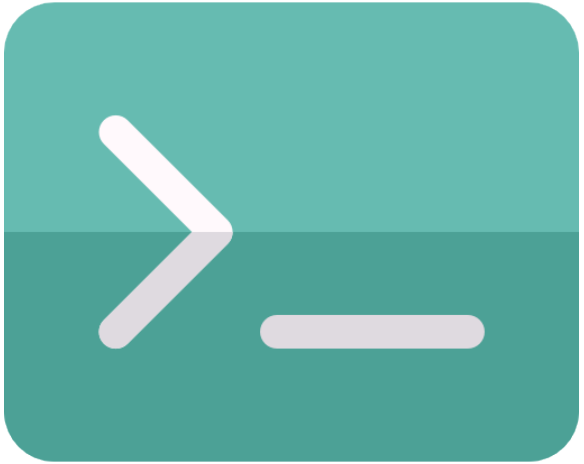
Final POC Link :

https://accounts.snapchat.com/accounts/sso?client_id=creativesuite-prod&referrer=https://snappublisher.snapchat.com/api/v1/media/5a75642e-e2b1-42f5-a8e4-a303251df20b/file/somthine.svg?%23pranav

BOUNTY

Reward : 7500$

CODE STEALING



1. What we do?
2. Secret Methodology
3. Case Study

CODE STEALING - What we do?

Our main AIM is to:

1. **Steal `authorization_code` of the application**
2. **Use it to login into user's account to user's account.**

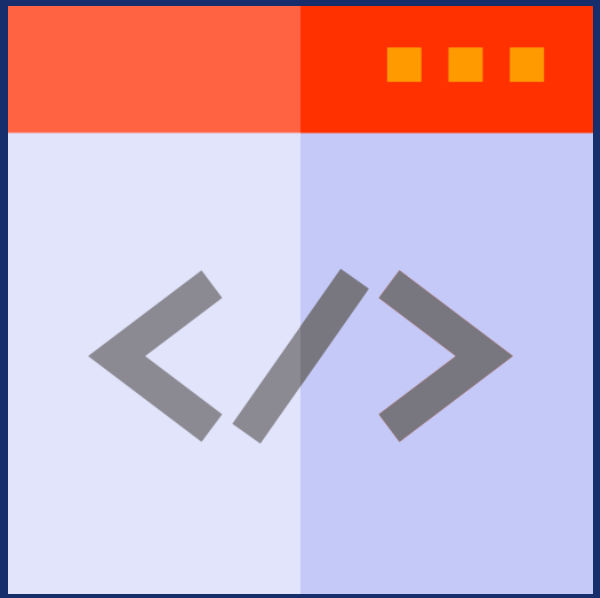
https://www.facebook.com/v2.7/dialog/oauth?response_type=code&display=popup&client_id=<client_id>&redirect_uri=<redirect_uri>&scope=email

CODE STEALING - What we do?

Exception:

1. `redirect_uri` while exchanging the `authorization_code` with `access_token` must match, when we got `authorization_code`.

CODE STEALING- Secret Methodology



1. Find domain used in `redirect_uri`.
2. Can you use `subdomains` in the `redirect_uri`.
3. Check if `authorization_code` dervied from manipulated `redirect_uri` works when fetching `access_token`.
4. Point the `redirect_uri` to a page
    1. XSS which can be used in `redirect_uri` to pass `authorization_code` to attacker.
    2. Subdomain takeover (allowed subdomain in `redirect_uri`)
    3. Loading user controlled external images, scripts, etc. (leaking via referrer)
5. Use the stolen `authorization_code` to login.

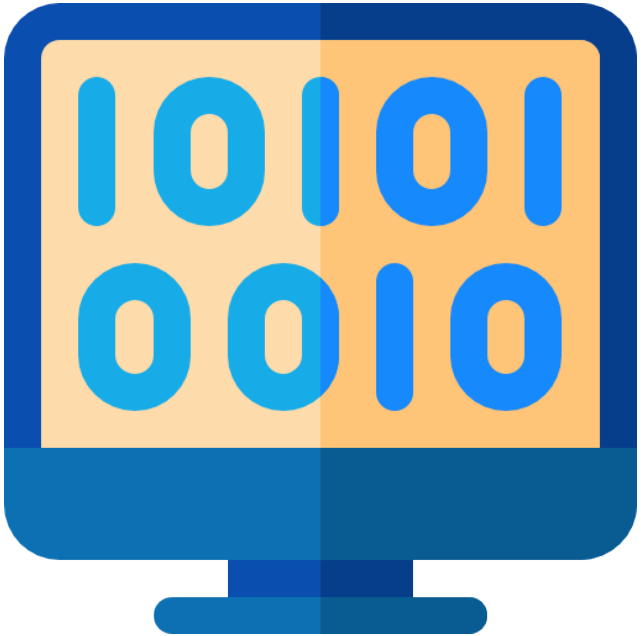
Case Study

Bug Name: Stealing login `authorization_code`

Program: Private

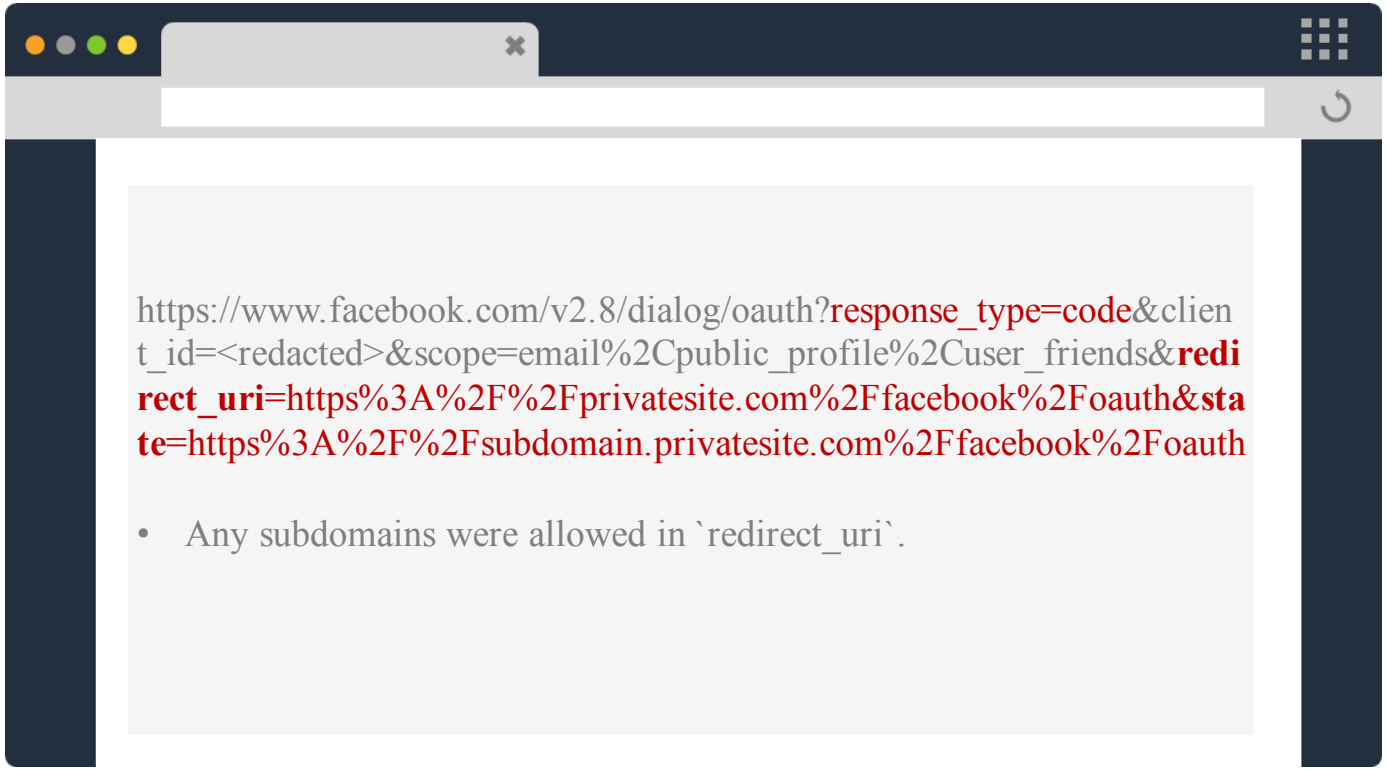
Found by: Pranav Hivarekar(@HivarekarPranav)

Case Study



a. Background information
    a. Company allowed registering subdomain of your choice. E.g. domain.privatesite.com
    b. Allowed customization by adding HTML, loading images, etc.
b. Login functionality used `Login in with Facebook`.
c. But used `authorization_code`.

PROOF OF CONCEPT



https://www.facebook.com/v2.8/dialog/oauth?response_type=code&client_id=<redacted>&scope=email%2Cpublic_profile%2Cuser_friends&**redirect_uri**=https%3A%2F%2Fprivatesite.com%2Ffacebook%2Foauth&**state**=https%3A%2F%2Fsubdomain.privatesite.com%2Ffacebook%2Foauth

- Any subdomains were allowed in `redirect_uri`.

PROOF OF CONCEPT

a. **Challenge**

Steal the `authorization_code` without manipulating `redirect_uri` of the Facebook app.

b. **Observation**

`state` param was used to redirect. So, I **could redirect to any subdomain.**

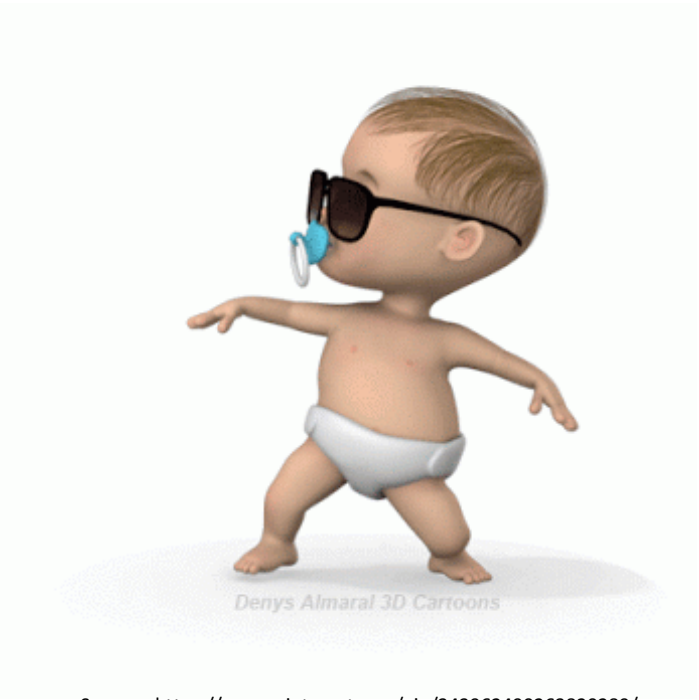
PROOF OF CONCEPT

a. Bought subdomain.privatesite.com

b. Hosted my own images and JS

PROOF OF CONCEPT

My reaction

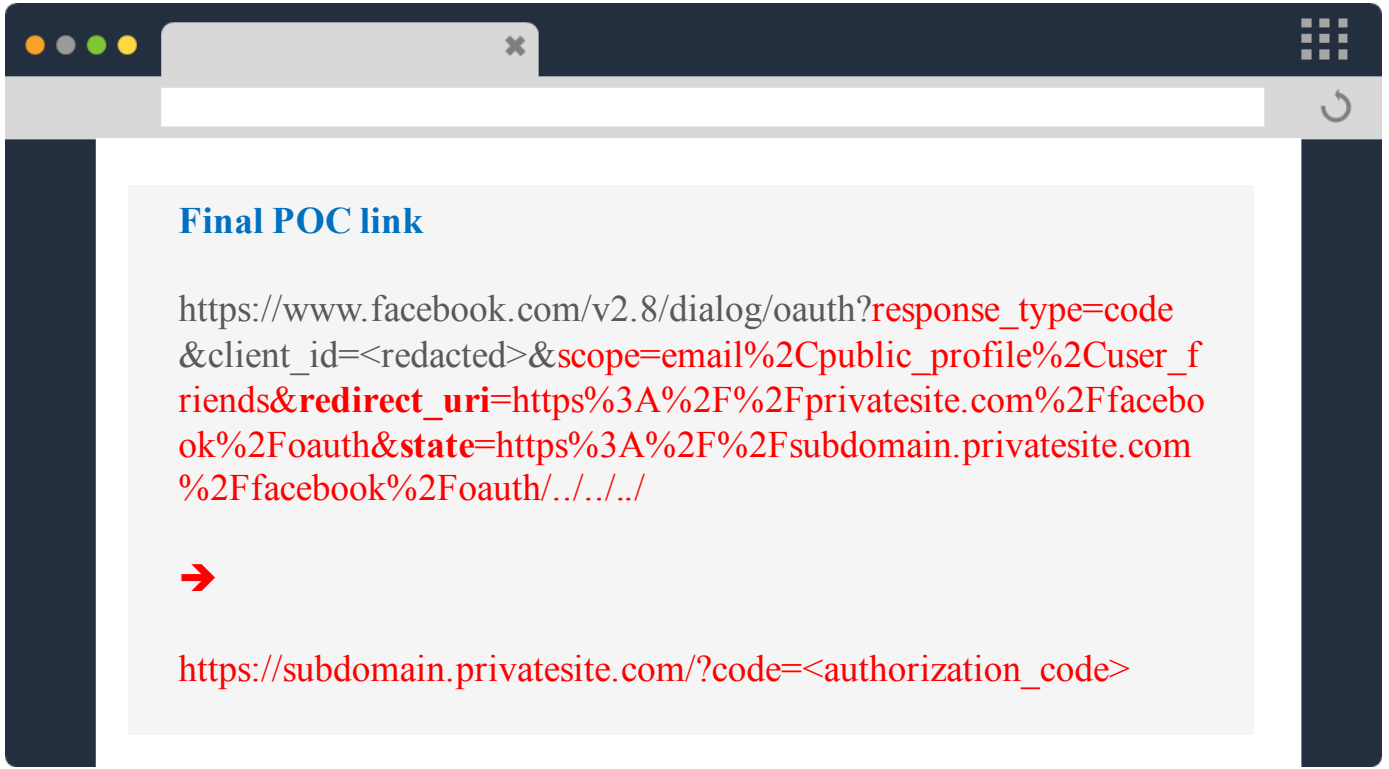


Source : https://www.pinterest.com/pin/343962490263820289/

ATTACK WORKFLOW

1. Victim is logged into privatesite.com via Facebook login.
2. Logout the victim via LOGOUT CSRF.
3. Send victim Facebook POC link.
4. This link sent `authorization_code` to subdomain under attacker's control.
5. Subdomain leaked the URL to images loaded from attacker's website via referrer header.
6. Attacker can then use the stolen `authorization_code` to login into victim's account.

PROOF OF CONCEPT



Final POC link

https://www.facebook.com/v2.8/dialog/oauth?response_type=code&client_id=<redacted>&scope=email%2Cpublic_profile%2Cuser_friends&**redirect_uri**=https%3A%2F%2Fprivatesite.com%2Ffacebook%2Foauth&**state**=https%3A%2F%2Fsubdomain.privatesite.com%2Ffacebook%2Foauth/../../../

➔

https://subdomain.privatesite.com/?code=<authorization_code>

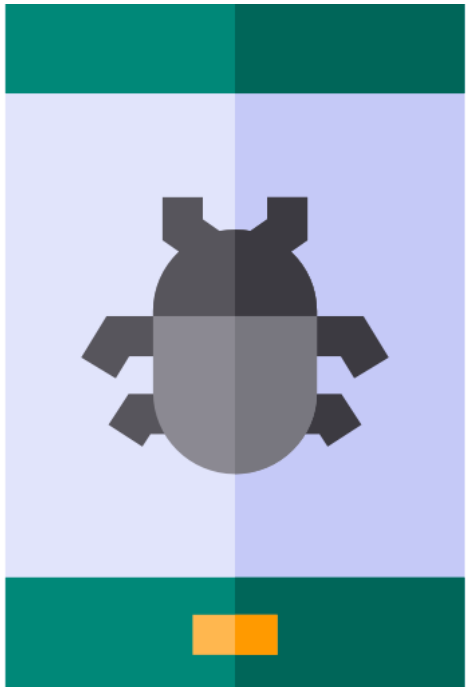
BOUNTY

Reward : 1000$

CSRF (MISSING `STATE` PARAM)

1. What we do?
2. Secret Methodology
3. Case Study

CSRF - What we do?



Our main AIM is to:

1. **Connect attacker's {Facebook} account to user's account.**
2. **Login via attacker's {Facebook} account into user's account.**

https://www.facebook.com/v2.3/dialog/oauth?response_type=code&display=popup&client_id=<client_id>&redirect_uri=<redirect_uri>&scope=email&state=<some_anti_csrf_token>

CODE STEALING - Secret Methodology

1. Check if `state` param in OAuth Authorization Link is validated?
2. Derive yourself a valid `authorization_code` link and do no use it.
3. Send this active `authorization_code` link to victim.
4. Your account will get connected with victim's account.
5. Now login via your own account.

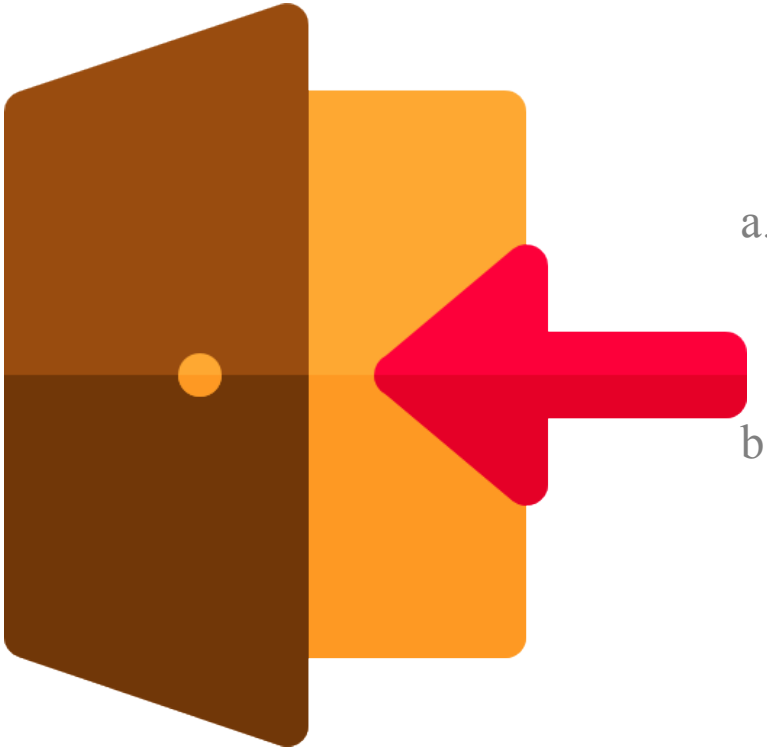
Case Study

Bug Name : Missing validation `state` param in Facebook OAuth allows to c onnect attacker's account to victim's account

Program: Private

Found by: Pranav Hivarekar(@HivarekarPranav)

Case Study

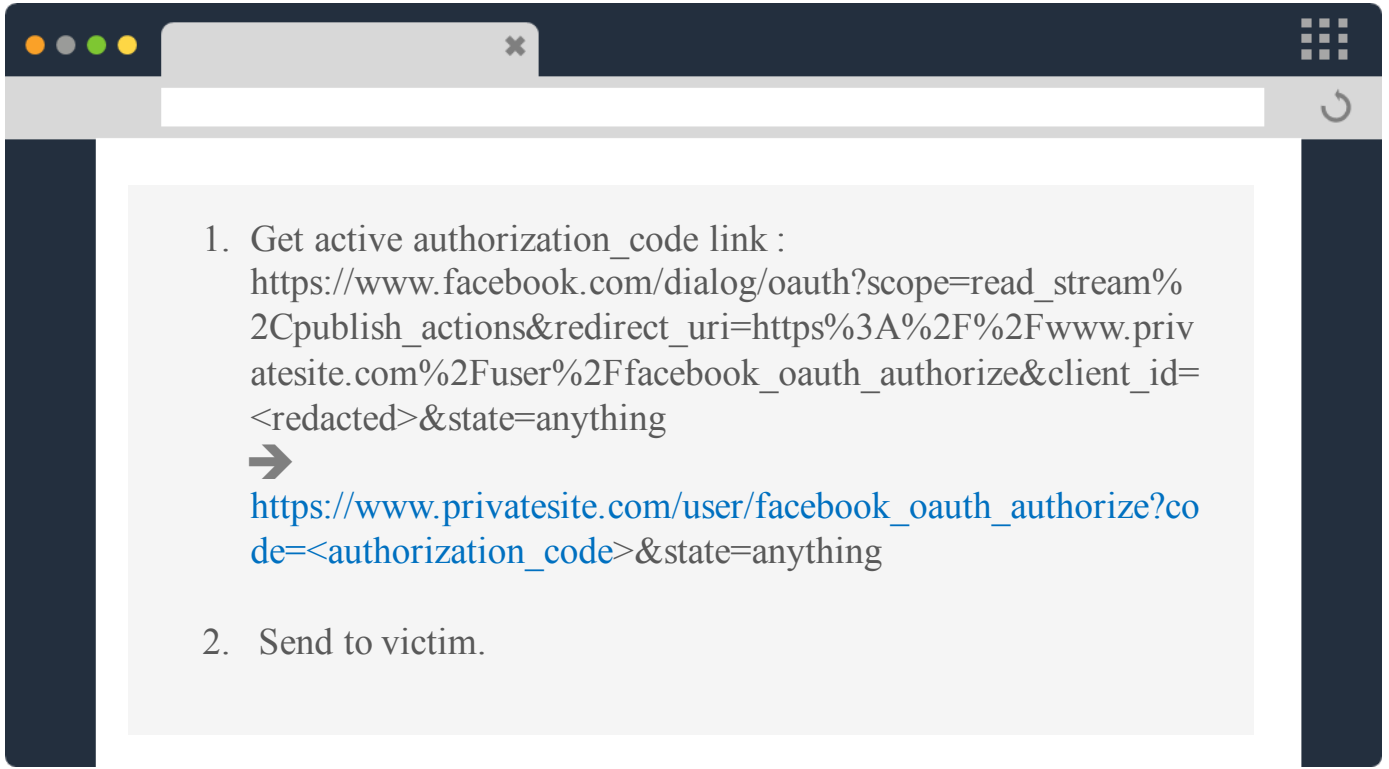


a. Background information
   a. Company allowed connecting Facebook account.
   b. Any content saved in the app will get auto-shared to Facebook.
b. But `state` param in the OAuth link never got validated.

ATTACK WORKFLOW

1. Attacker initiates `Connect Facebook` flow.
2. Attacker derives a valid `authorization_code` link and doesn't use it.
3. Victim is logged into privatesite.com
4. Attacker sends active `authorization_code` link to the victim.
5. Attacker's Facebook account is connected to victim's account.
6. Any content saved/posted in account will get shared to attacker's Facebook account. (Access to private content)

PROOF OF CONCEPT



1. Get active authorization_code link :
   https://www.facebook.com/dialog/oauth?scope=read_stream%2Cpublish_actions&redirect_uri=https%3A%2F%2Fwww.privatesite.com%2Fuser%2Ffacebook_oauth_authorize&client_id=<redacted>&state=anything
   ➔
   https://www.privatesite.com/user/facebook_oauth_authorize?code=<authorization_code>&state=anything

2. Send to victim.

BOUNTY

Duplicate

Source : https://www.askideas.com/wp-content/uploads/2016/11/Funny-Sad-Baby-Face-Photo.jpg

TOKEN IMPERSONATION

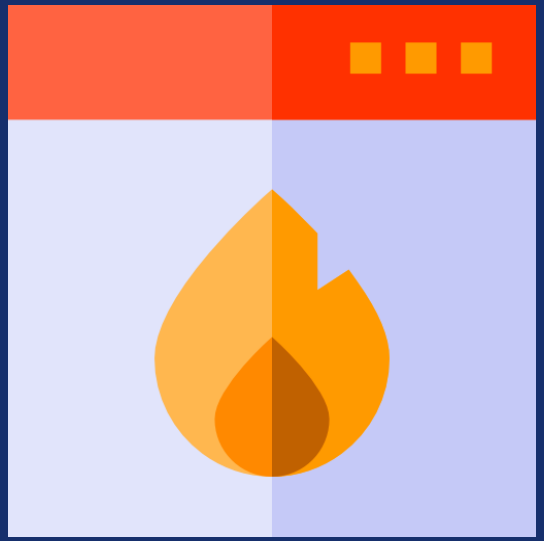
1. What we do?
2. Secret Methodology
3. Case Study

TOKEN IMPERSONATION - What we do?

Our main AIM is to:

1. Use victim's valid `access_token` from any third party app.
2. Pass `access_token` of the victim from different app and get unauthorized access to the account.

TOKEN IMPERSONATION- Secret Methodology



1. Check if login endpoint accepts `access_token` directly.
2. Derive an `access_token` from a different app.
3. Pass it on to the login endpoint.

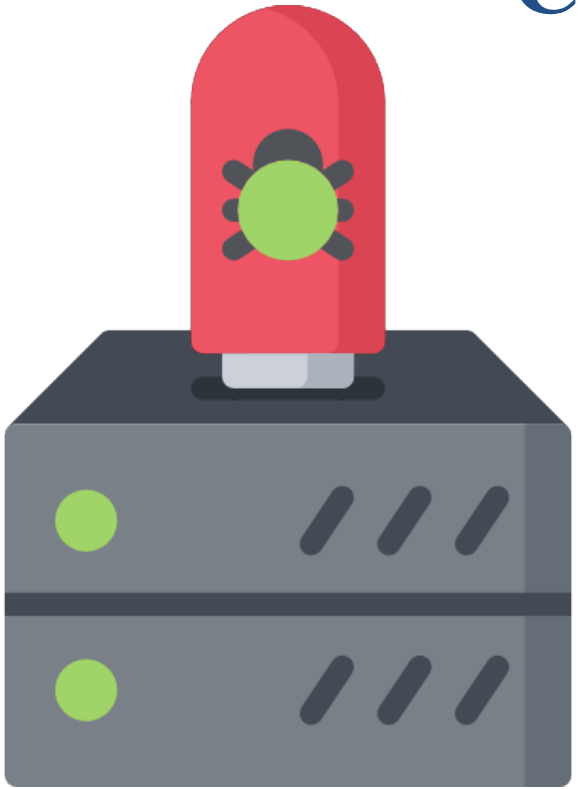
Case Study

Bug Name : Authentication Bypass - Using Any Valid `access_token` to Login

Program: Private

Found by: Pranav Hivarekar(@HivarekarPranav)

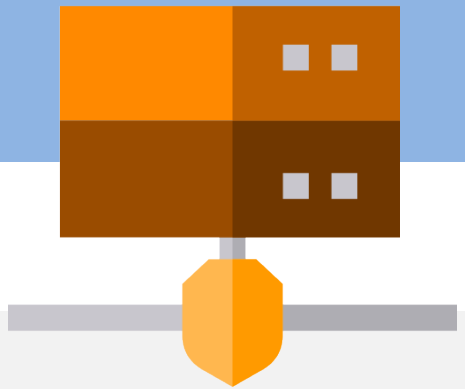
Case Study



a. Background information
    a. Company used `Login with Facebook` account.
    b. Used `access_tokens` to login.

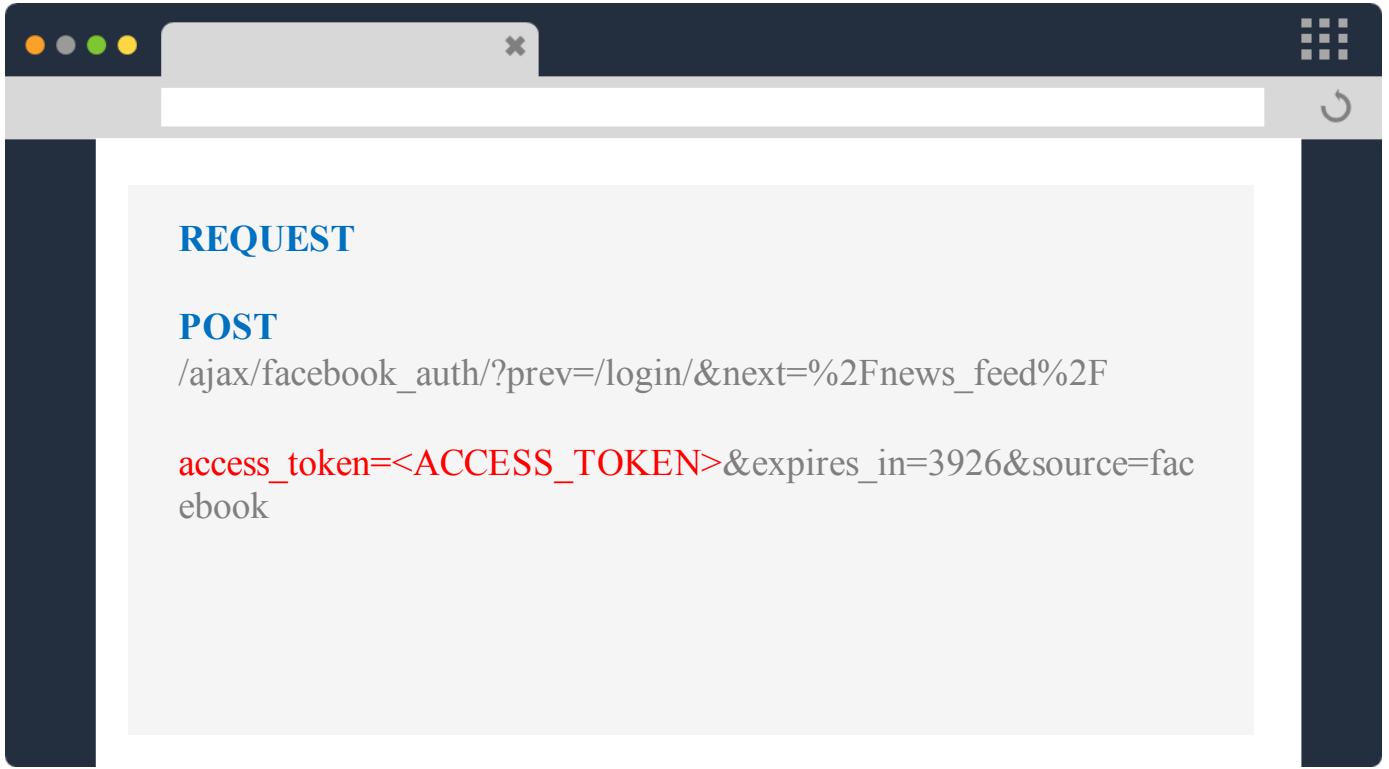
ATTACK WORKFLOW

Attacker derives `access_token` for the user from a different app.



Uses it to bypass authentication.

PROOF OF CONCEPT



REQUEST

POST
/ajax/facebook_auth/?prev=/login/&next=%2Fnews_feed%2F

access_token=<ACCESS_TOKEN>&expires_in=3926&source=facebook

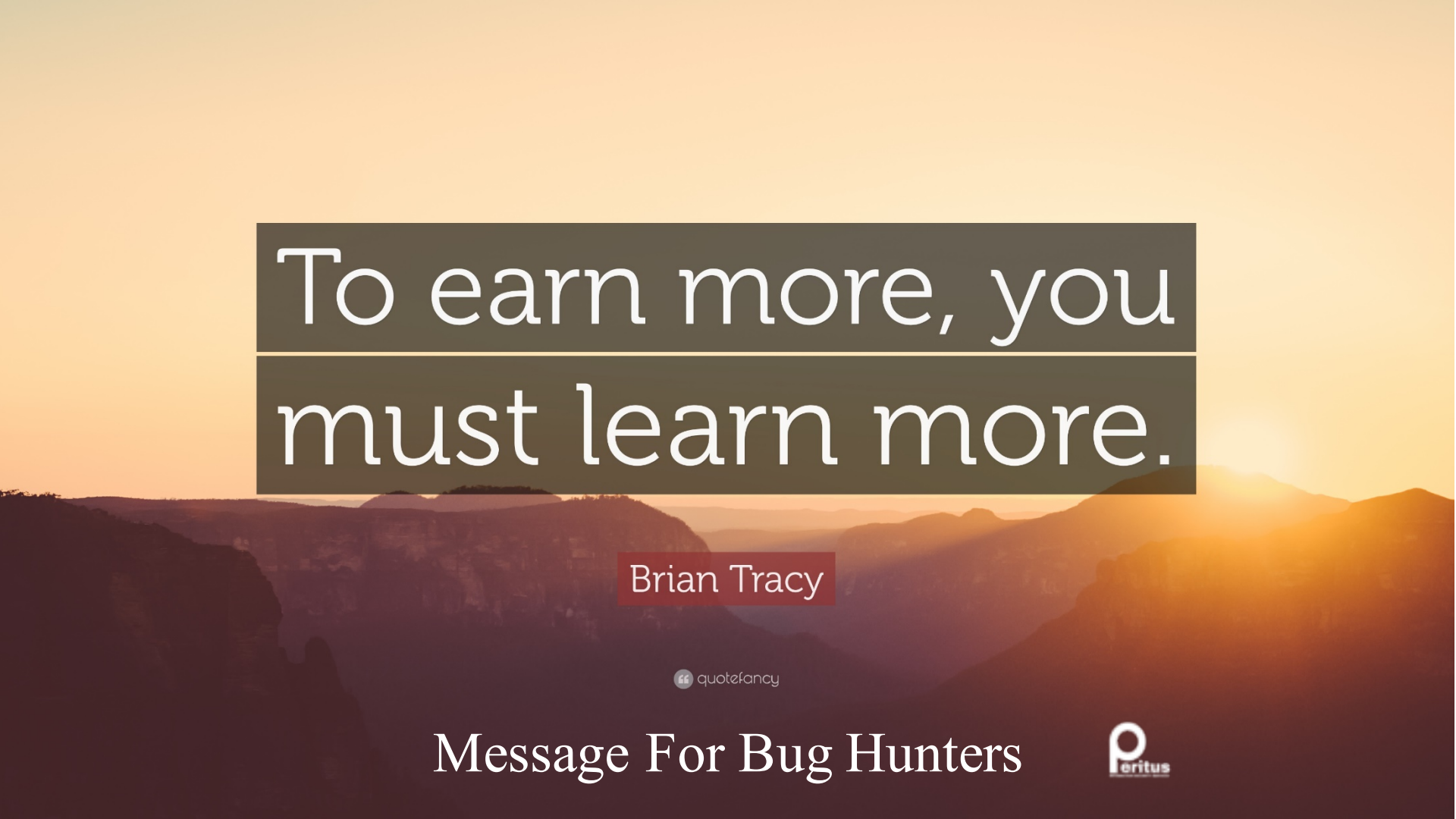
BOUNTY

Reward : 1000$

CONCLUSION

3 secret techniques to use when pentesting third party integrations on a website:

1. **Token/Code Stealing**
2. **CSRF (missing `state` param)**
3. **Token Impersonation**



Message For Bug Hunters

THANKS

Bugcrowd

Audience

Check out

→https://training.peritusinfosec.com

Further Reading

Token/Code Stealing

https://whitton.io/articles/uber-turning-self-xss-into-good-xss/
https://stephensclafani.com/2017/03/21/stealing-messenger-com-login-nonces/
https://philippeharewood.com/swiping-facebook-official-access-tokens/
https://whitton.io/articles/obtaining-tokens-outlook-office-azure-account/
https://www.arneswinnen.net/2017/06/authentication-bypass-on-ubers-sso-via-subdomain-takeover/
http://blog.intothesymmetry.com/2016/11/all-your-paypal-tokens-belong-to-me.html
https://pranavhivarekar.in/2015/01/29/twitters-bug-importing-contacts-oauth-flaw/
https://ngailong.wordpress.com/2017/08/07/how-i-could-steal-your-google-bug-hunter-account-with-two-clicks-in-ie/
https://ngailong.wordpress.com/2017/08/07/uber-login-csrf-open-redirect-account-takeover/
https://ngailong.wordpress.com/2017/08/29/one-more-thing-to-check-for-sso-flickr-ato/
https://ngailong.wordpress.com/2017/11/22/uber-redirect_uri-is-difficult-to-do-it-right/
https://pranavhivarekar.in/2015/01/29/twitters-bug-importing-contacts-oauth-flaw/

Further Reading

CSRF

https://hackerone.com/reports/127703
https://www.josipfranjkovic.com/blog/hacking-facebook-oculus-integration-csrf
https://www.josipfranjkovic.com/blog/hacking-facebook-csrf-device-login-flow

Token Impersonation

https://medium.freecodecamp.org/hacking-tinder-accounts-using-facebook-accountkit-d5cc813340d1

References

https://en.wikipedia.org/wiki/OpenID
https://en.wikipedia.org/wiki/OAuth
https://tools.ietf.org/html/rfc6749
https://oauth.net/


eritus
INFORMATION SECURITY SERVICES


bugcrowd
OUTHACK THEM ALL™