RSAConference2020

San Francisco | February 24 – 28 | Moscone Center

HUMAN ELEMENT

SESSION ID: MBS1-F03

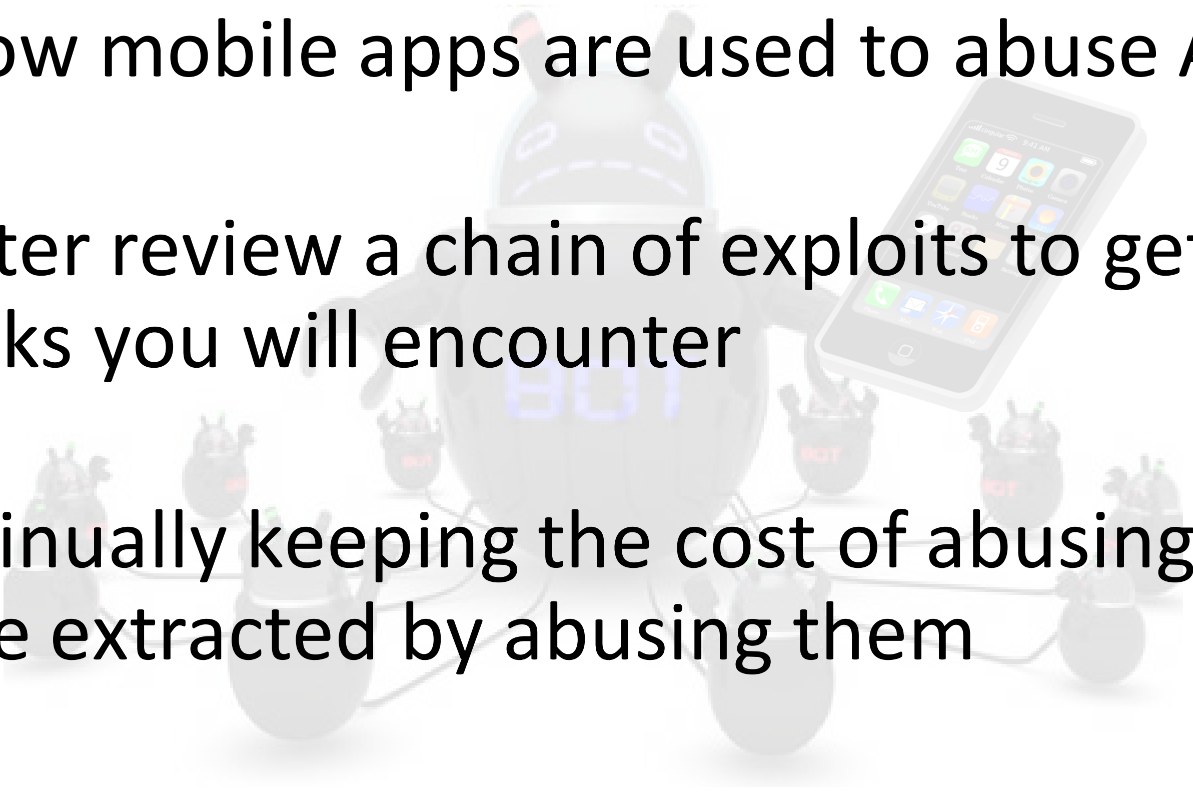
API Abuse through Mobile Apps: New Attacks, New Defenses

Skip Hovsmith

Principal Engineer
CriticalBlue
@SkipHovsmith

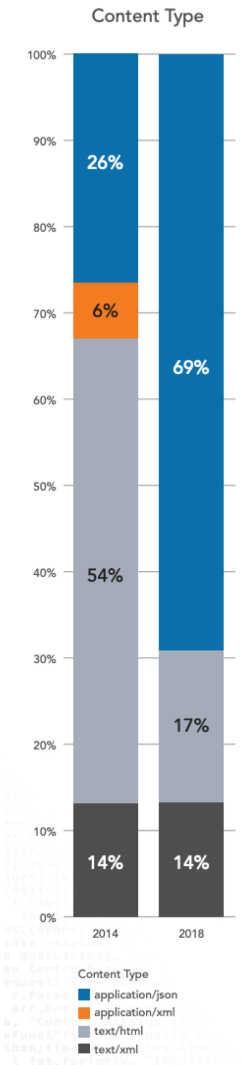
#RSAC

Apply What You Learn Today

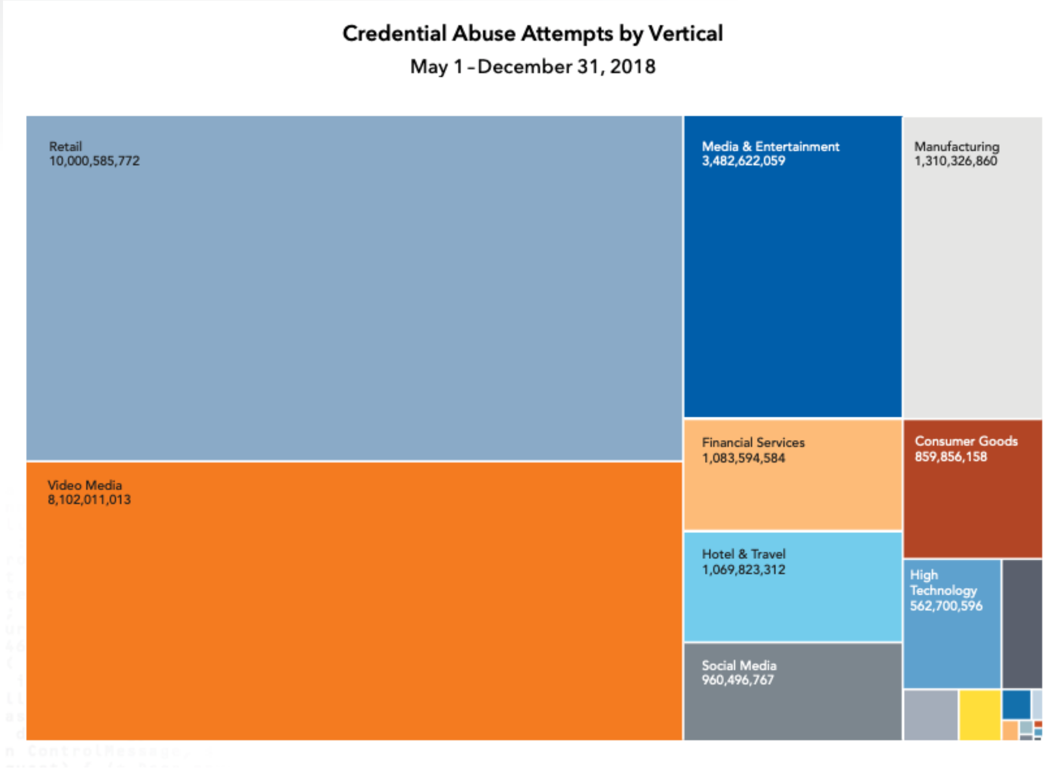


- Appreciate how mobile apps are used to abuse APIs
- Follow and later review a chain of exploits to get a feel for the types of attacks you will encounter
- Invest in continually keeping the cost of abusing your APIs higher than the value extracted by abusing them

The Dark API Economy

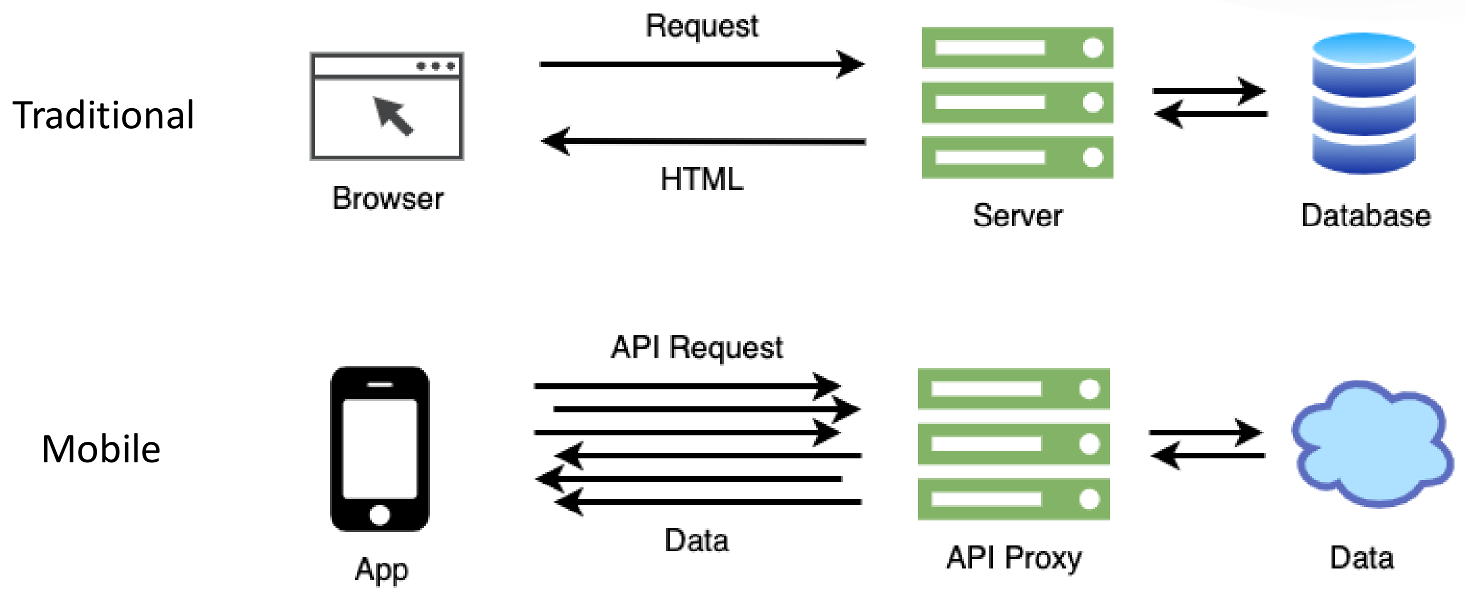


- In 2018, Akamai observed:
  - 83% of CDN traffic was API content, not HTML.
  - Over 27B credential abuse attempts in 6 months

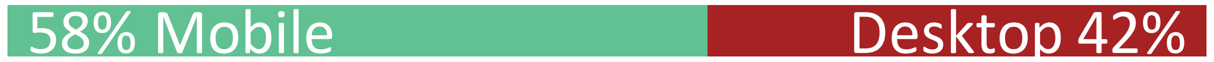


- Gartner reports:
  - By 2022, API abuses will be the most-frequent attack vector resulting in data breaches for enterprise web applications.

Mobile Apps Rely on APIs



- Shift from presentation markup to raw data transfer
- Stateless API calls are great for attackers



API Abuse in the Mobile Market

1. Exploit a mobile app and channel to architect an API attack

2a. Use bots to launch high volume API-driven attacks:

- Fast or sustained data exfiltration
- Account takeover attacks
- Application-level denial of service attacks

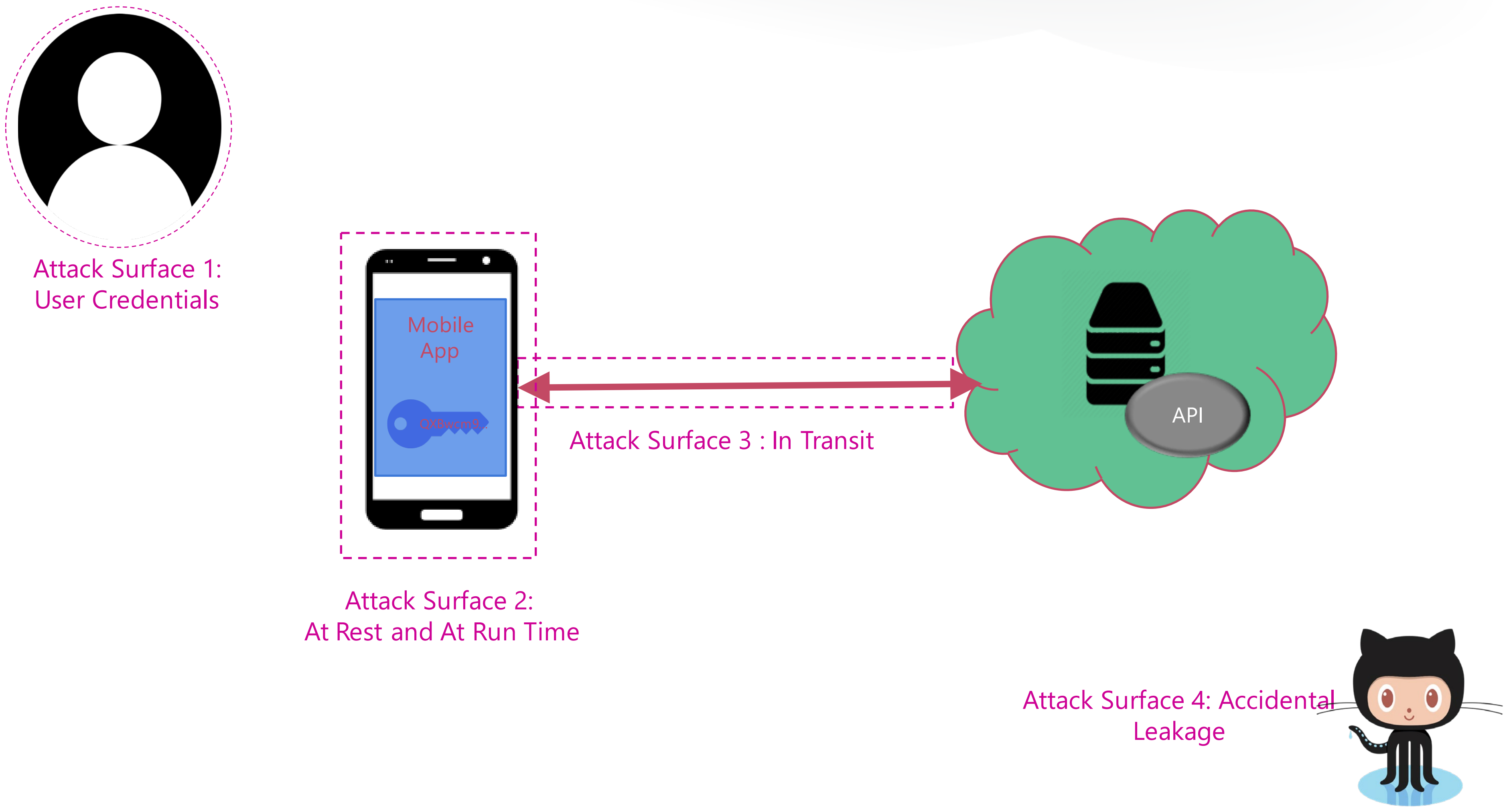
2b. Use tampered apps to game the implicit API business model

- Modify API call sequences for gain or frustration

API Abuse Defense Objectives

- Prevent API reverse engineering
- Make it hard to construct a valid API call
- Make it hard enough that it's not worth it

Mobile Attack Surfaces



OWASP Security Risks

Mobile Top Ten

M1: Improper Platform Usage

M2:Insecure Data Storage

M3: Insecure Communication

M4: Insecure Authentication

M5: Insufficient Cryptography

M6: Insecure Authorization

M7: Client Code Quality

M8: Code Tampering

M9: Reverse Engineering

M10: Extraneous Functionality

Enable

API Top Ten

A1: Broken Object Level Access Control

A2: Broken Authentication

A3: Improper Data Filtering

A4: Lack of Resources and Rate Limiting

A5: Missing Fun/Resource Access Control

A6: Mass Assignment

A7: Security Misconfiguration

A8: Injection

A9: Improper Assets Management

A10: Insufficient Logging and Monitoring

RSAConference2020

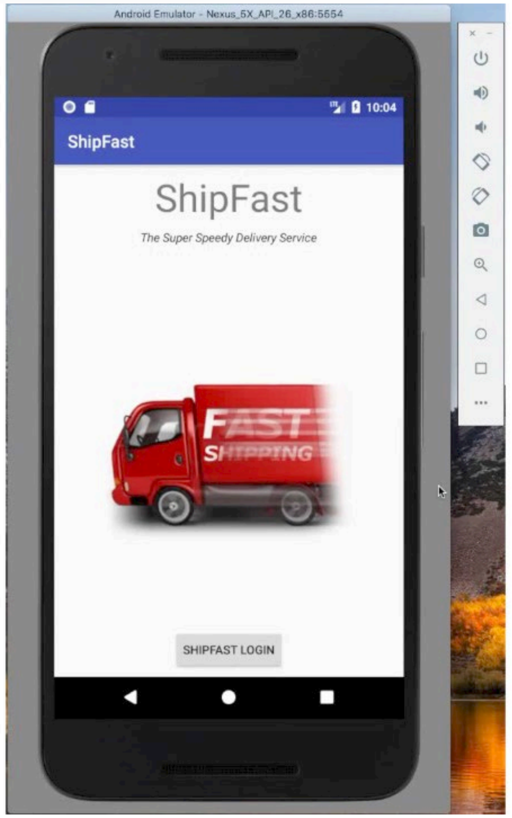
ShipFast and ShipRaider

A Hypothetical Package Delivery Service

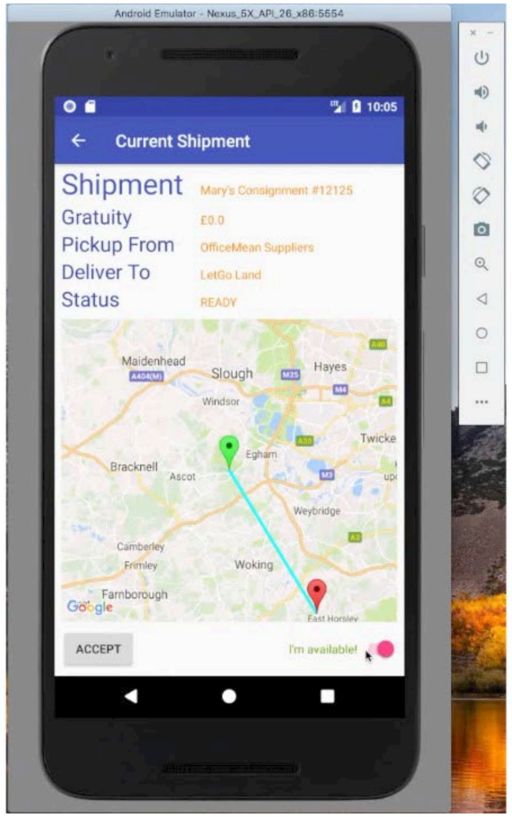
The ShipFast Platform

- ShipFast Driver's App (React Native)
- ShipFast REST API
- API Gateway
- ShipFast API Services
- Authentication Services

- Public Repo: *tbd*


Android Emulator - Nexus_5X_API_26_x86:5554
10:04
ShipFast
ShipFast
The Super Speedy Delivery Service
FAST SHIPPING
SHIPFAST LOGIN

The ShipFast Driver's App



Android Emulator - Nexus_5X_API_26_x86:5554

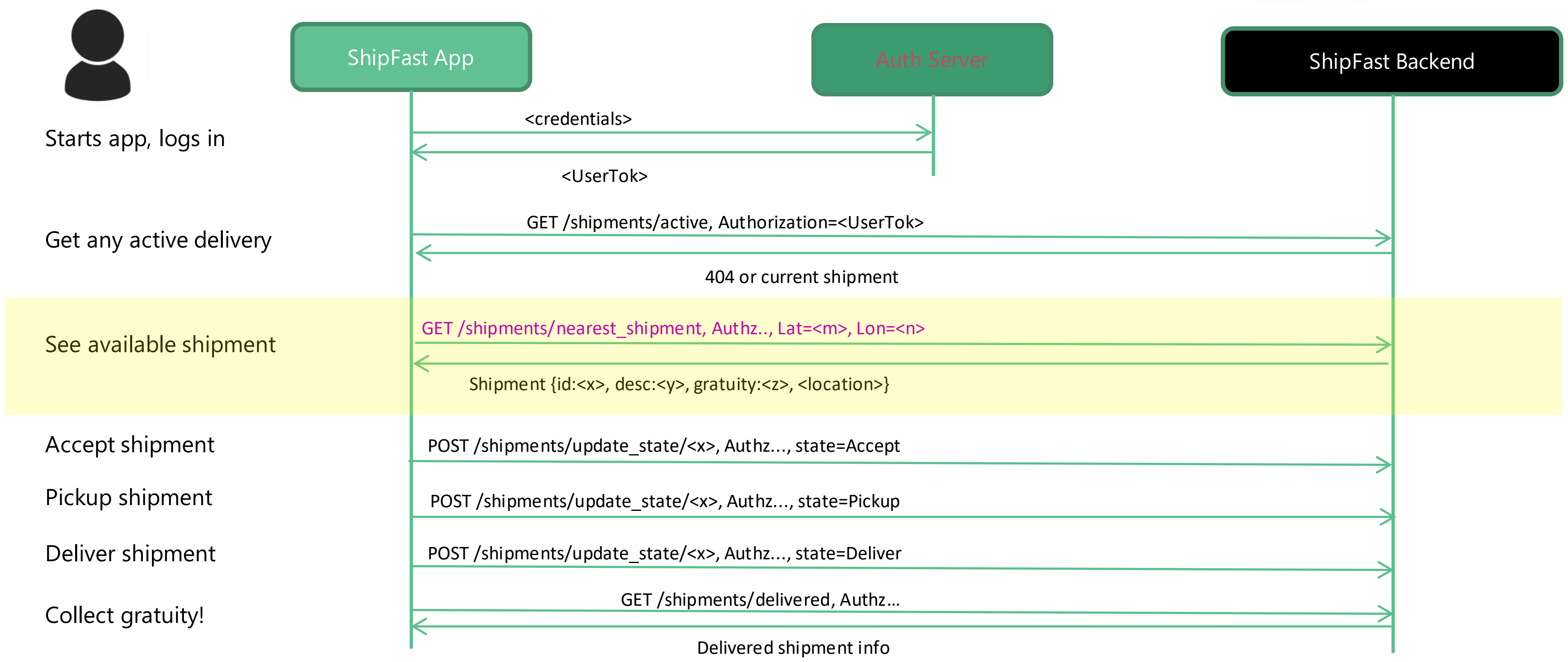
10:05

Current Shipment

Shipment	Mary's Consignment #12125
Gratuity	£0.0
Pickup From	OfficeMean Suppliers
Deliver To	LetGo Land
Status	READY

- Driver assigned nearest shipment
- Driver paid by distance driven and pre-established gratuity

API Sequence for Typical Package Delivery

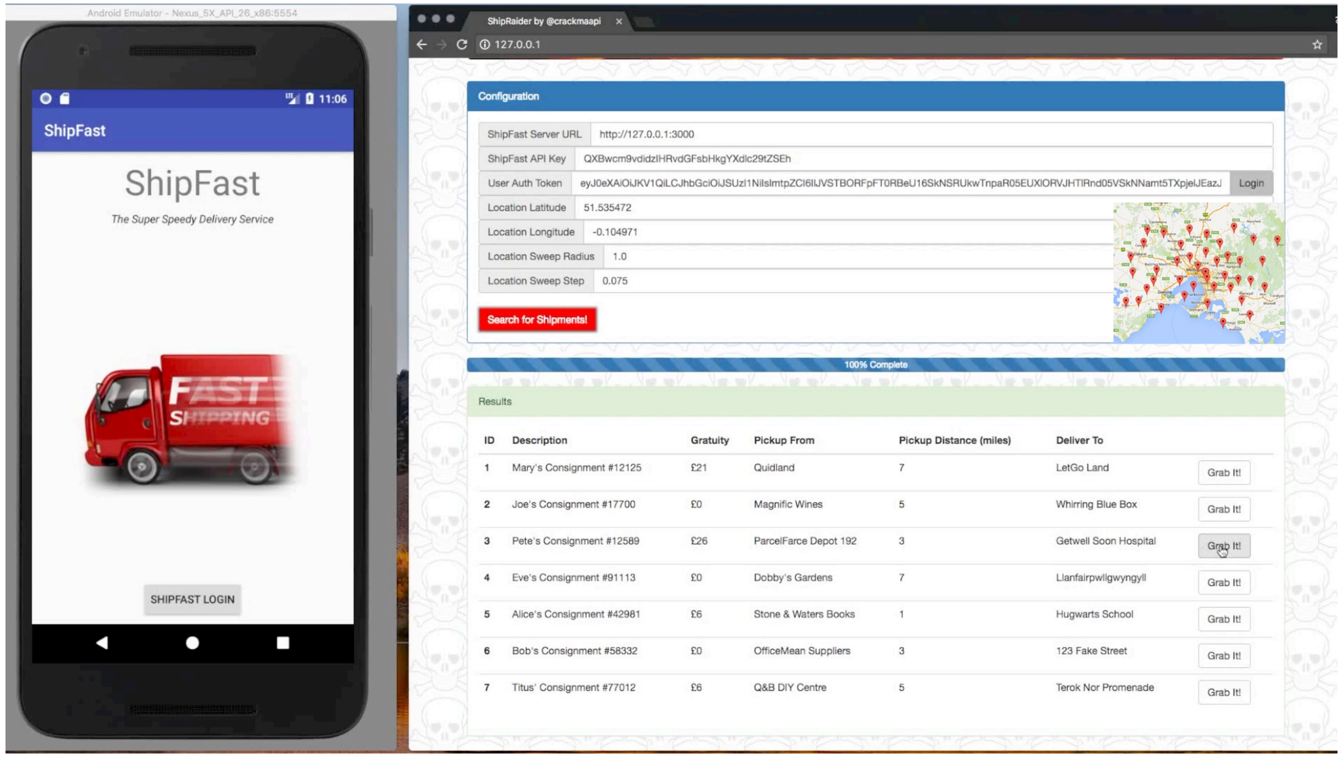

ShipFast App
Auth Server
ShipFast Backend
Starts app, logs in
<credentials>
<UserTok>
Get any active delivery
GET /shipments/active, Authorization=<UserTok>
404 or current shipment
See available shipment
GET /shipments/nearest_shipment, Authz.., Lat=<m>, Lon=<n>
Shipment {id:<x>, desc:<y>, gratuity:<z>, <location>}
Accept shipment
POST /shipments/update_state/<x>, Authz..., state=Accept
Pickup shipment
POST /shipments/update_state/<x>, Authz..., state=Pickup
Deliver shipment
POST /shipments/update_state/<x>, Authz..., state=Deliver
Collect gratuity!
GET /shipments/delivered, Authz...
Delivered shipment info

In Headers:

Authorization: Bearer <access-token>

SF_API_Key: <api-key>

The ShipRaider Driver’s Assistant


Android Emulator - Nexus_5X_API_26_x86:5554
11:06
ShipFast
ShipFast
The Super Speedy Delivery Service
FAST SHIPPING
SHIPFAST LOGIN
ShipRaider by @crackmaapi
127.0.0.1
Login
Search for Shipments!
100% Complete

Configuration

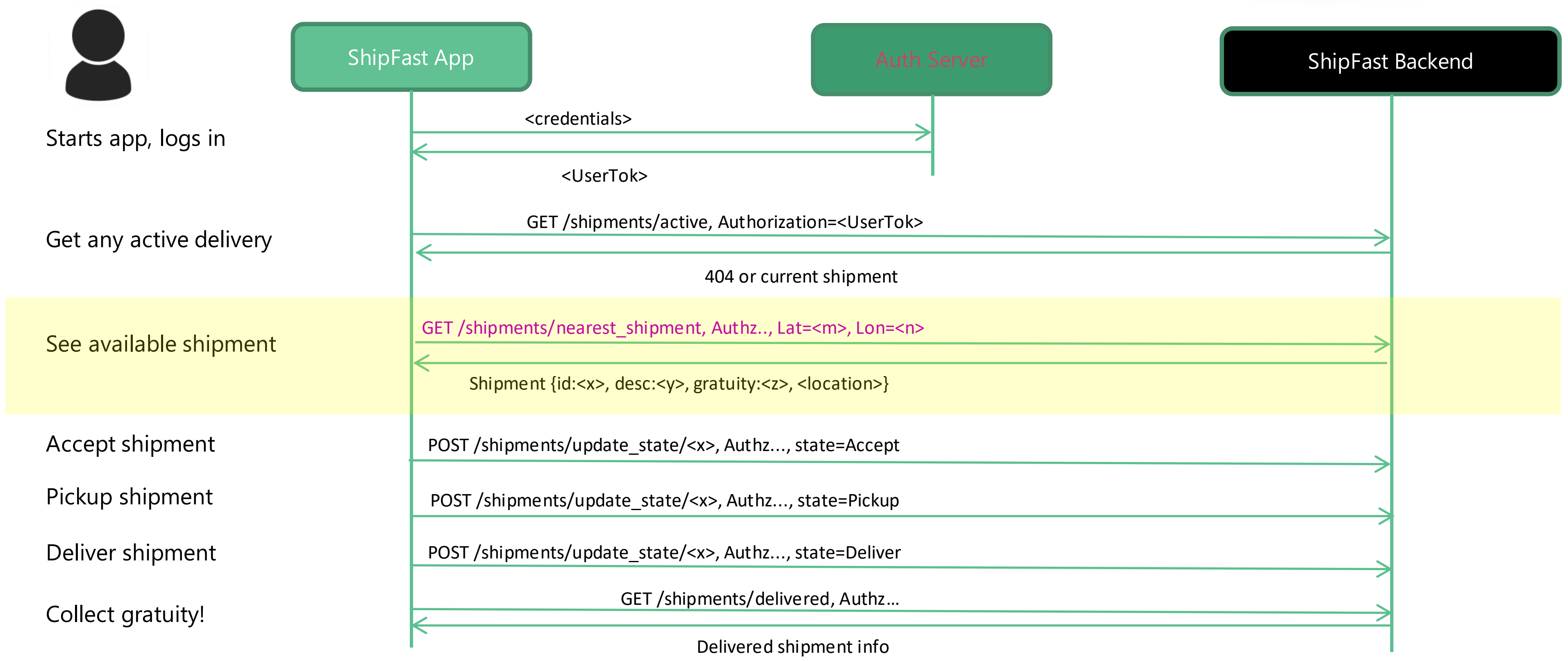
ShipFast Server URL	http://127.0.0.1:3000
ShipFast API Key	QXBwcm9vdidzlHRvdGFsbHkgYXdlc29tZSEh
User Auth Token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImtpZCI6IlJVSTBORFpFT0RBeU16SkNSRUkwTnpaR05EUXlORVJHTlRnd05VSkNNamt5TXpjelJEazJ
Location Latitude	51.535472
Location Longitude	-0.104971
Location Sweep Radius	1.0
Location Sweep Step	0.075

Results

ID	Description	Gratuity	Pickup From	Pickup Distance (miles)	Deliver To	
1	Mary's Consignment #12125	£21	Quidland	7	LetGo Land	Grab It!
2	Joe's Consignment #17700	£0	Magnific Wines	5	Whirring Blue Box	Grab It!
3	Pete's Consignment #12589	£26	ParcelFarce Depot 192	3	Getwell Soon Hospital	Grab It!
4	Eve's Consignment #91113	£0	Dobby's Gardens	7	Llanfairpwllgwyngyll	Grab It!
5	Alice's Consignment #42981	£6	Stone & Waters Books	1	Hugwarts School	Grab It!
6	Bob's Consignment #58332	£0	OfficeMean Suppliers	3	123 Fake Street	Grab It!
7	Titus' Consignment #77012	£6	Q&B DIY Centre	5	Terok Nor Promenade	Grab It!

- Raider selects highest gratuity from nearby deliveries

API Sequence for Driver's Exploit



Starts app, logs in
<credentials>
<UserTok>

Get any active delivery
GET /shipments/active, Authorization=<UserTok>
404 or current shipment

See available shipment
GET /shipments/nearest_shipment, Authz.., Lat=<m>, Lon=<n>
Shipment {id:<x>, desc:<y>, gratuity:<z>, <location>}

Accept shipment
POST /shipments/update_state/<x>, Authz..., state=Accept

Pickup shipment
POST /shipments/update_state/<x>, Authz..., state=Pickup

Deliver shipment
POST /shipments/update_state/<x>, Authz..., state=Deliver

Collect gratuity!
GET /shipments/delivered, Authz...
Delivered shipment info

In Headers:
Authorization: Bearer <access-token>
SF_API_Key: <api-key>

RSAConference2020

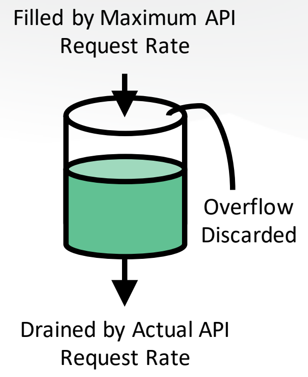
ShipFast Security Evolution

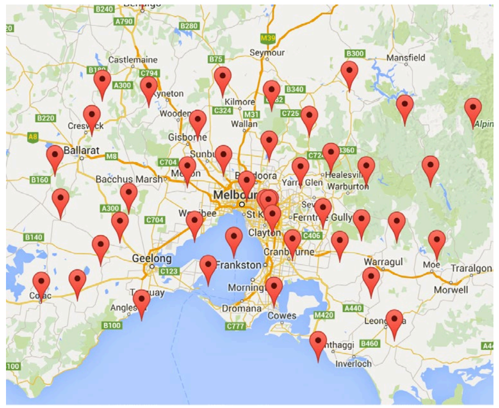
Initial Security Posture

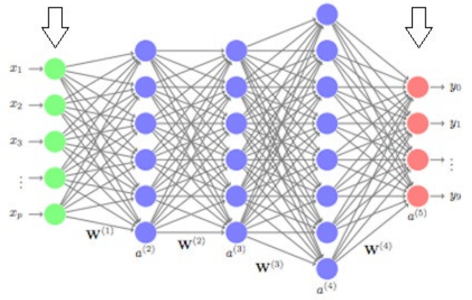
- OAuth2 Authorization Flow
- Static API Key in Code Bundle
- API calls over HTTPS

Common Back-End Defenses

- Rate limiting
- Data constraint checking
- Calling pattern anomalies


Filled by Maximum API Request Rate
Overflow Discarded
Drained by Actual API Request Rate

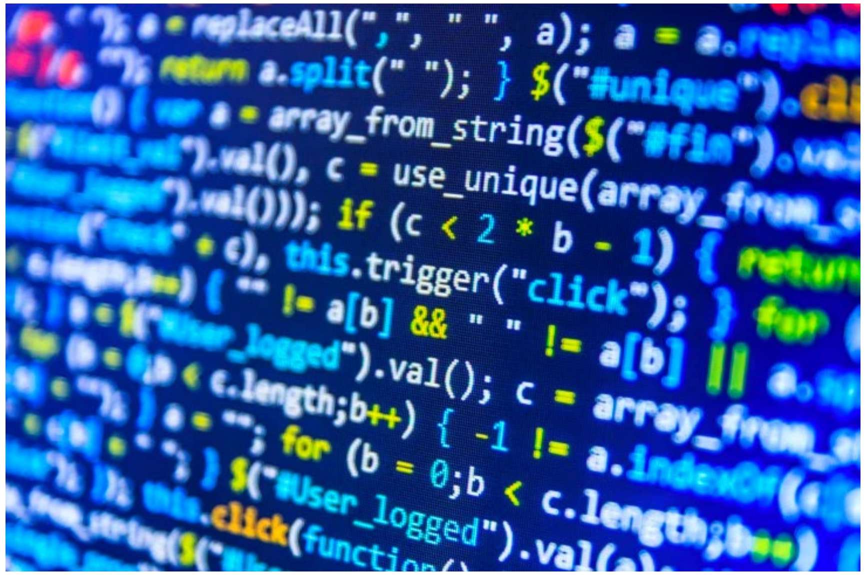




Assume we can beat these statistical checks

Attack by Inspection

- Attacker unzips app package
- Inspects index.android.bundle
- Finds API call fetches in code
- Finds API key in code
- User volunteers credentials
- Clones app with gratuity scanning

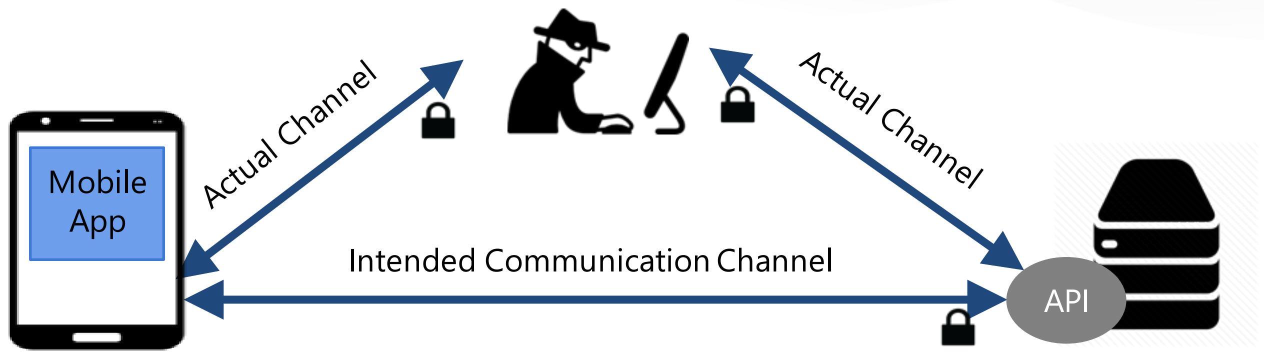


Defend Through Obfuscation

- Obfuscate calling logic
- Obfuscate API calling & key strings
- Don’t roll your own
- Do block debugging

- For RN, use:
  – https://github.com/javascript-obfuscator/javascript-obfuscator

Attack using Man-in-the-Middle



- Insert custom certificate in device trust store
- Show MitM proxy attack steps
- Easy to observe and modify API requests & responses

Defend by Pinning Channel

- Client keeps whitelist of trusted certificates
- Only accepts connections from a whitelisted certificate
- Attacker cannot match a whitelisted certificate or know the certificate’s private key
- Use react-native-cert-pinner

Attack by Unpinning Channel

- Use an instrumentation framework to hook the pinning decision function

FRIDA

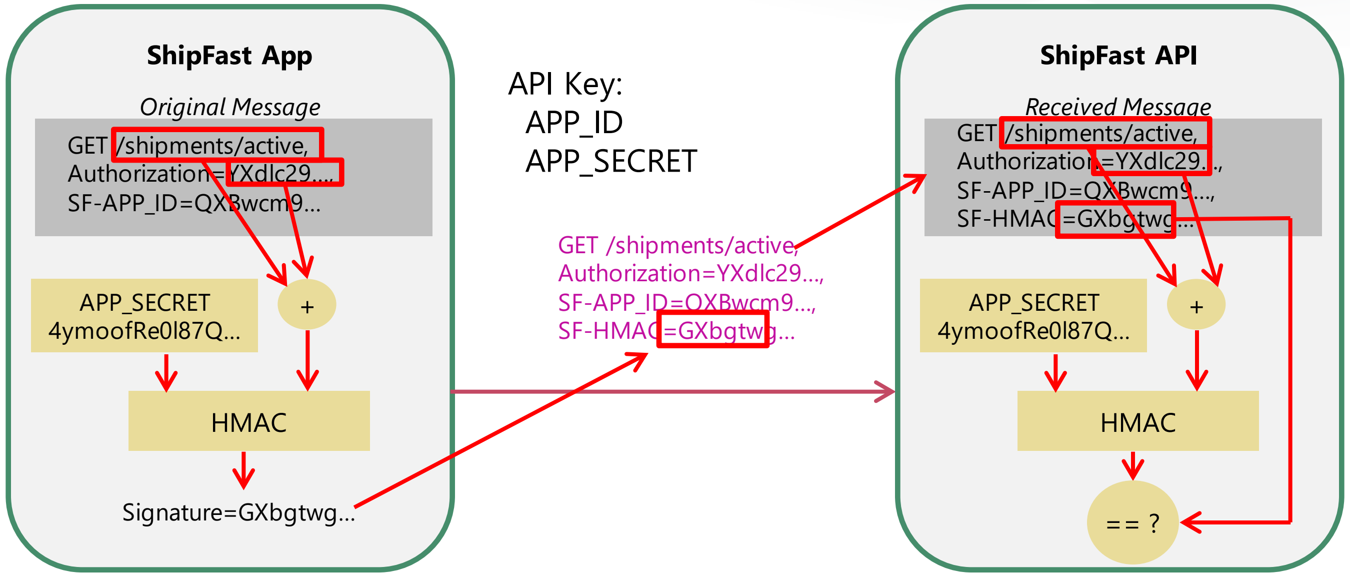
Defend by Blocking Instrumentation

- Block rooting and block hooking
- Change API key and/or API version!

Defeat by Product Manager

- No Pinning!
  - Server certificates, their public keys or fingerprints are client secrets
  - Certificates may expire or be revoked, bricking the app
  - Updating the certificates on the client is a maintenance challenge

Defend by App-Level Message Protection



- Assume secret hidden somehow inside app
- Signing proves client possesses secret and request is untampered
- Secret not transmitted; only run time signature
- Responses can be signed; can use full encryption

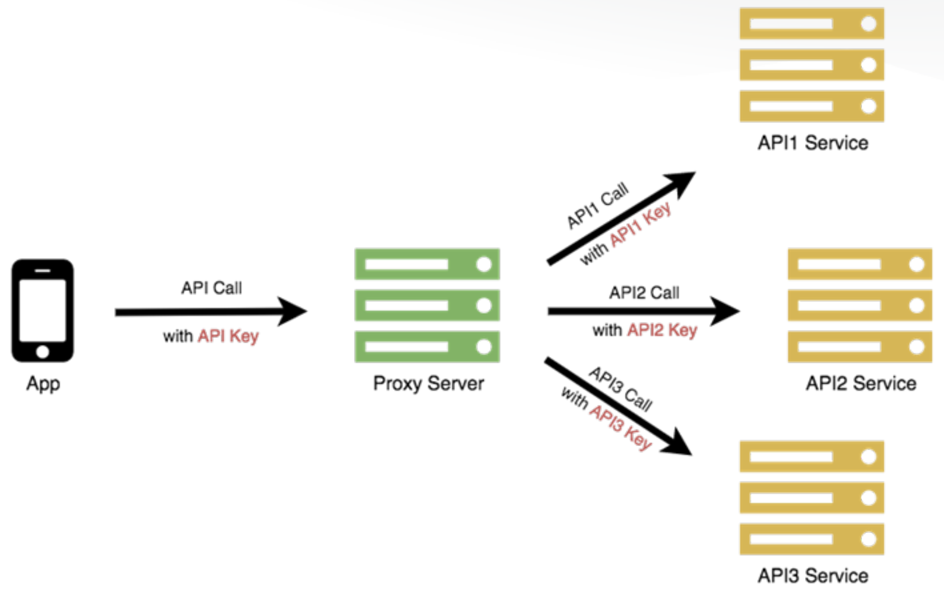
Defend by Removing API Key from App Source

- Download the API Key assuming Trust On First Use (TOFU)
- Store Key in secure storage (keystore/keychain)
- Use https://www.npmjs.com/package/react-native-secure-key-store

Attack by Finding HMAC Pieces

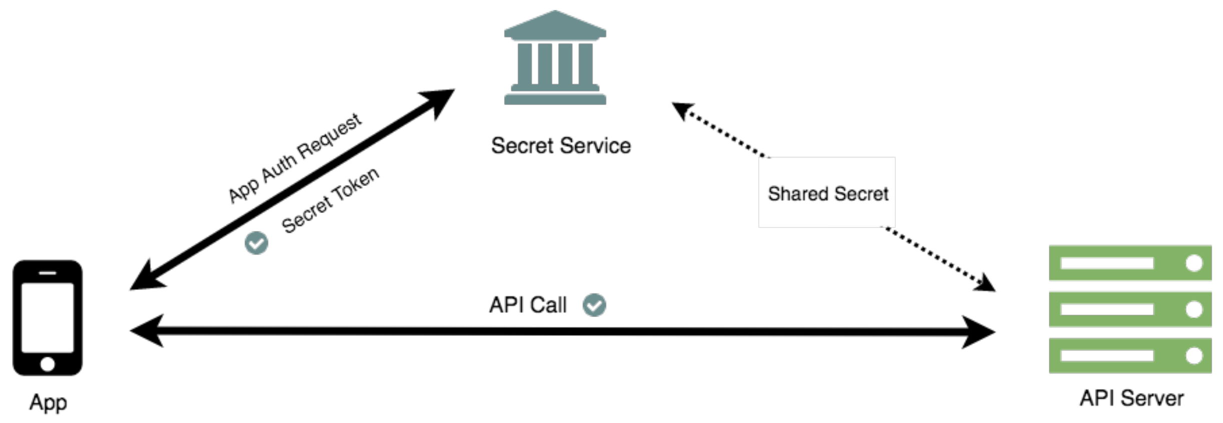
- Use MitM to inspect API calls and find HMAC header
- Guess HMAC algorithm – HMAC-SHA256?
- Root phone and Inspect app's data stores
- Debug to find HMAC string in memory

Defend by Adding API Proxy



- Define app-specific API between app & service
- Move 3rd party APIs and their API key insecurities to behind proxy server

Defend by using Secrets Service



- Move secret from the app to a secrets service
- App receives a signed, short-lived JWT token on request
- Secret can be revoked or updated without touching app

How Does App Authenticate to Secrets Service?

- User authentication not good enough
- Remotely attest code not tampered
  - Reliably perform non-replayable dynamic app integrity measurements
  - The app does not make or know the integrity decision
- Verify security checks are in place (not rooted, not debugged, not emulator)
- Prototype by verifying package signature

Defend by Reintroducing Pinning Service

- Securely grab pinning certificate from secrets service at app start up
- Not strictly necessary to update API key or version as the key was never seen in the app or the channel
- Add this to react-native-certificate-pinner package

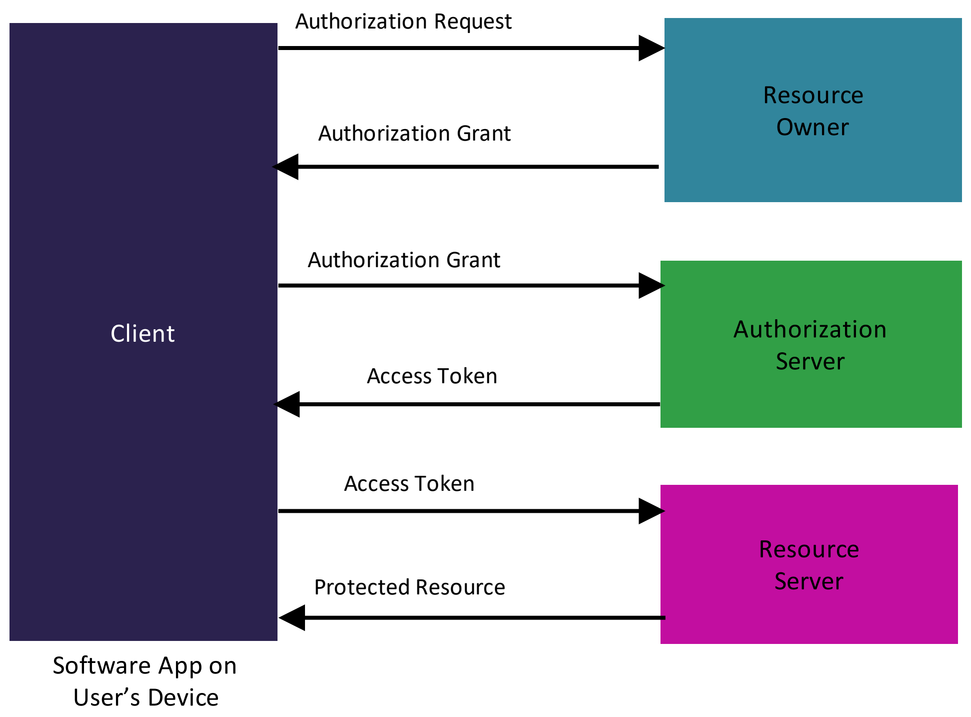
Attacker Pivots to a Less Secure App

RUNNING SHOES? THAT'S STUPID! YOU CAN'T OUT-RUN A BEAR!
TRUE, BUT I ONLY NEED TO OUTRUN YOU!

http://philjulianoillustration.com/comic/2010-04-28-bear-joke/

Authentication

OAuth2 Authorization Flow

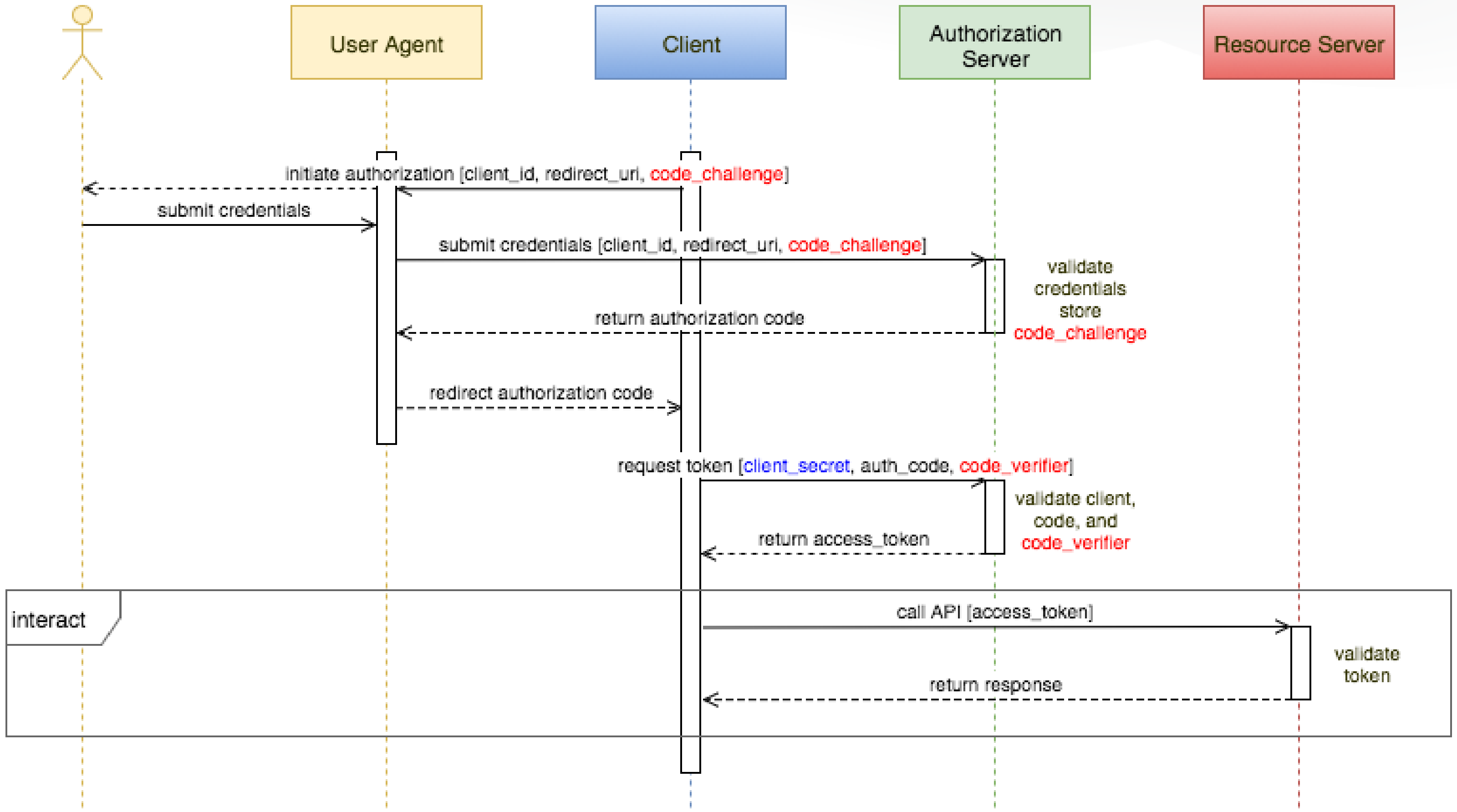


- Resource Owner is typically the user
- Consents to authorization scope

- Verifies Resource Owner identity
- Issues tokens for access

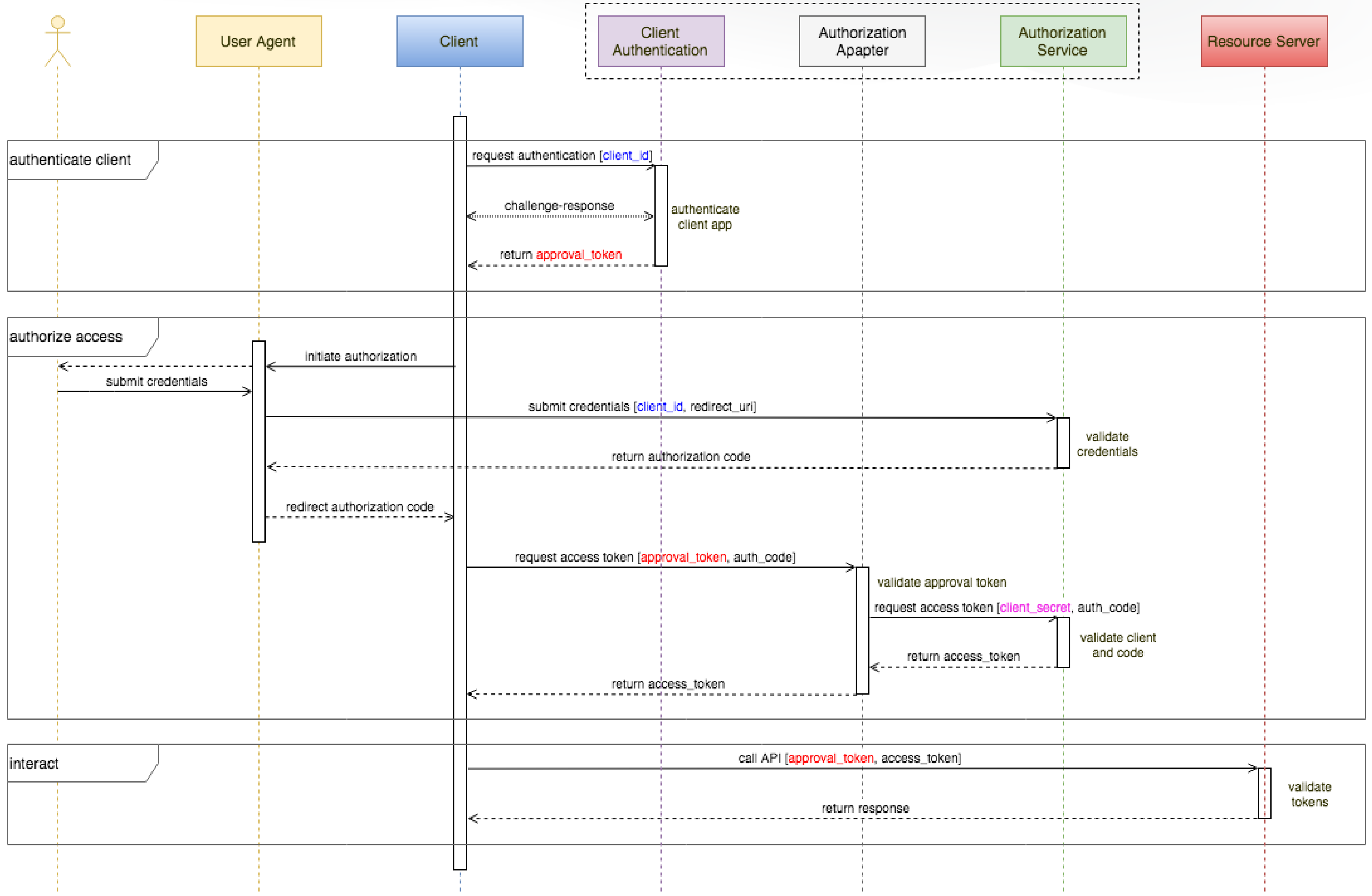
- Holds the protected user resources
- The API backend that provides content

Defend using PKCE


User Agent
Client
Authorization Server
Resource Server
initiate authorization [client_id, redirect_uri, code_challenge]
submit credentials
submit credentials [client_id, redirect_uri, code_challenge]
validate credentials store code_challenge
return authorization code
redirect authorization code
request token [client_secret, auth_code, code_verifier]
validate client, code, and code_verifier
return access_token
interact
call API [access_token]
validate token
return response

- On mobile, prevent malicious party from intercepting authorization code
- Code challenge is hash of random value
- Mitigates against leaky client_secret
- Server compares with hash of code_verifier

Strengthen Oauth2 Using Secrets Service



- Prevent fake apps from authenticating

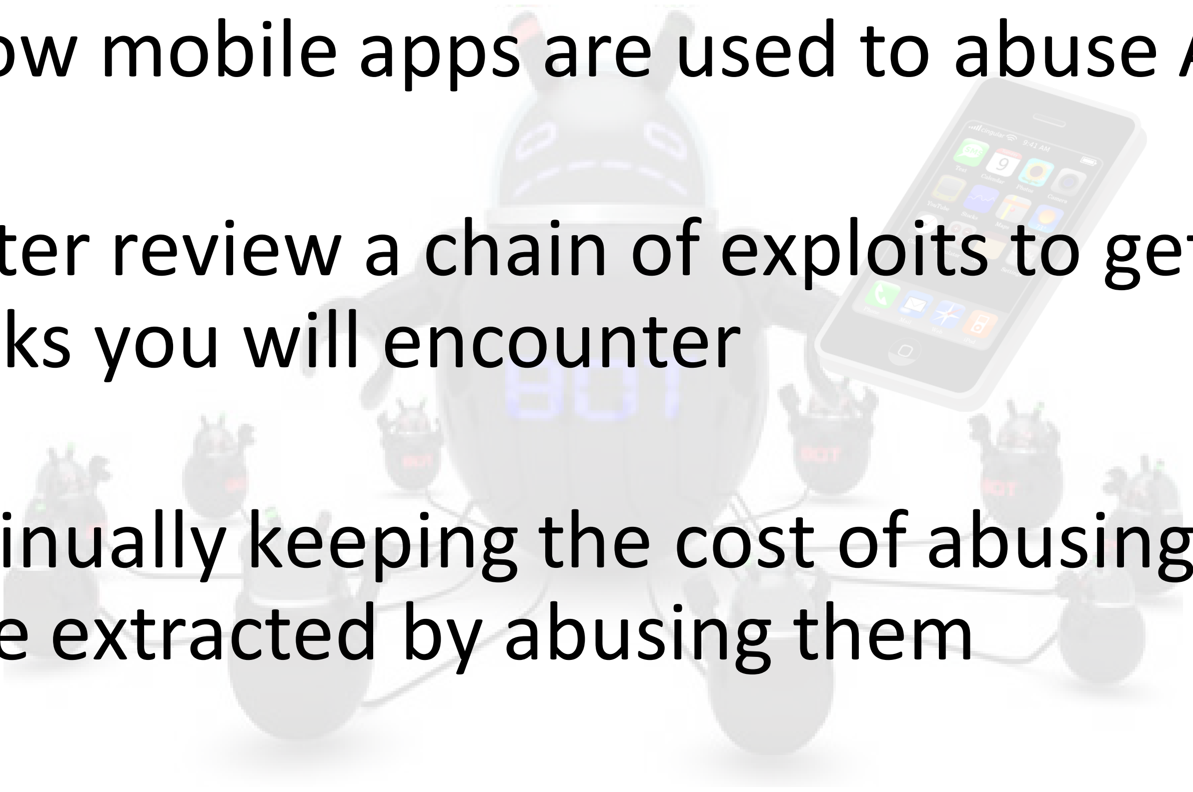
Authorization in Context

- Decide API authorization from multiple signals
  - User authentication
  - App authentication
  - Channel authentication
  - Device authentication
  - Behavioral profiles (time of day, location)
  - Mobile Captchas (accelerometer, touch)

API Abuse Defense Objectives

- Prevent API reverse engineering
- Make it hard to construct a valid API call
- Make it hard enough that it's not worth it

Apply What You Learn Today



- Appreciate how mobile apps are used to abuse APIs
- Follow and later review a chain of exploits to get a feel for the types of attacks you will encounter
- Invest in continually keeping the cost of abusing your APIs higher than the value extracted by abusing them