# Rogue Mobile App is mother of all threats

## - An Uncontrollable Lethal Threat


**A rogue mobile that steals authentication data, can be created in just a few hours and distributed to the masses through social media, making millions of users permanent victims.**

**It can bypass all security checks. Using stolen authentication data, fraudulent transactions can be repeatedly carried out without organizations or users noticing them.**

Authentication of all apps, of all organizations, is written exactly the same way. So, the same stealing process can be repeated for others.

Users become permanent victims after installing a rogue app. Using this stolen authentication data, billions of dollars can be siphoned out from user accounts in a short period, repetitively.

***In a typical banking example, if a thief could push a rogue app on just one million users' phones and the average balance of a user's account is just $1,000, then the thief could siphon out $1 billion in a few hours.***

# The Base Vulnerability - API Exploitation


- All mobile applications communicate with servers using APIs.
- An API server merely processes a payload, without understanding the authenticity of the received data.
- If data could be managed at the client-level, then there would be zero security.
- It is very simple to manage data if either the data or the application is static, which is always the case in mobile applications.
- Rogue apps essentially exploit this shortcoming to carry out fraudulent transactions.

A transactional mobile app merely provides a user interface. It collects user inputs, device data, and sends them to a server using an API call. The actual process is carried out by its server. Hence, it is very simple to create a rogue app in days from scratch exploiting APIs. Details of APIs and their payload can be obtained from network trace externally without tracing the app code.

# Why Rogue App is Lethal?

- Rogue mobile apps that exploit APIs can be created in a few hours.
- Rogue App can be distributed to the masses through social media.
- Once a user downloads and uses it once, he/she becomes a permanent victim.
- It is impossible for an organization to know who has downloaded a rogue app.
- Using stolen credentials, hacker can repeatedly transact from user's account without organization and user realizing fraud.
- Using custom programs exploiting APIs, thieves can transact from millions of user accounts in a short period.

# Creating Rogue App - A Simple Process

## Android

- Just download the APK from Playstore and de-compile it using apktool.
- Modify either DEX code or Java code. Put in a few lines of code that traps authentication data and sends it back to the thief using an API call or add code to carry-out fraudulent transactions.
- Rename the package, sign it with own signature, and publish it on Playstore as Private.
- Share the download link with persuasive messages on social media.
- Users may download this app and forward the message to others, reaching billions of people in just a few minutes.
- These rogue apps are fully functional apps that send data to a server.

## iOS

**Can't decompile iOS app? Don't worry.**

- Convert an Android rogue app to iOS using tools like MechDome or just create a new iOS app using the same APIs.
- Change the request header to an Android based user-agent.
- The server will treat it as an Android App.
- With sideloading of iOS apps allowed, the threat level and ease of distribution, iOS app will be the same as Android app.

# Breaking Anti-Tampering Check

Currently, anti-tampering checks are the only method to ensure an app is not modified.

- This is carried out by comparing either the hash/checksum value of app signature or program object of the running app with the objects of the original app..
- Hash-value/checksum value is also a static data, they can be obtained externally from the original app without modifying the original app,
- These obtained values can be substituted in the appropriate variable of .the rogue app to behave it as the original app, thus bypassing anti-tampering checks.
- Even C++ based anti-tampering code can be bypassed by modifying the de-assembled code or underlying java classes.

  Browse https://www.cybernetsecurityinc.com/presentation/Breaking_Security_Checks.pdf to see how each and every check can be bypassed.

# Bypassing Security Protection

## Certificate-Pinning

- Certificate pinning creates a trust between the application with the server during network call.
- In case of successful trusting, the data is encrypted using public-private key, thus giving protection from man-in-middle attacks.
- But all certificates and corresponding passwords are always hardcoded in the application. Thus, they can be easily obtained from the decompiled original application.

## App Monitoring Service

- App monitoring service tries to identify rogue app on Playstore.
- Unfortunately, it cannot identify private apps. Also, it takes many days to identify. Meanwhile, millions of users would have already installed rogue app.

## OTP

- In case of OTP 2FA, a rogue app that is installed on the user's mobile, can read the OTP from the received SMS on user's phone and then call confirmation API passing this OTP value or send it to thief, which is further submitted while calling confirmation API.

## Encryption

- Apps always use symmetric encryption where the encryption password is static and hardcoded.
- One can always get this encryption password from the original app. Use it to decrypt data, modify it, and resubmit to carry out fraudulent transactions.

## Biometric

- It is a myth that biometrics provide any protection.
- A biometric value in an app is nothing but a string.
- It can always be stolen and substituted like any other data.
- Rather, it is worse as it does not change/reset in user's life time.

# Mobile Security Solution Space

**Currently there are three types of solutions in mobile security space:**

## MDM/End Point

The primary objective is to detect malicious apps on a user's phone. They cannot stop any rogue app exploiting APIs

## App Hardening

These solutions encrypt java classes. This encrypted code is always available as cleartext code after the first run. So, a rogue app can be created using API from the decrypted code.

## SDK-Based

An SDK is introduced as part of an app. It produces only static data. Like any other data, static data produced from this SDK can be stolen or managed. So, it does not provide protection from rogue app. Also, it can be simply bypassed.

**SecureChannel,** using virtualization technology is the only known solution that can provide strong protection from rogue apps. Apart from this, it protects from all other threats prevented by other solutions.

# Our Proprietary Virtualization Technology

**The SecureChannel plugin** introduced in mobile applications creates dynamic virtual images of business data, authentication data, app data, device data, and network data. These images change every one second. These images are private and are relative to the internal properties of the device, the network and the app. They are one-time use, short-lived and can be submitted from the generating device only. These images are created using mobile time-based encryption keys. Neither time nor encryption keys are ever shared with the SecureChannel server.

The SecureChannel server can calculate the exact time this virtual image was generated on a user's phone. This dynamic image is processed by SecureChannel cloud-based de-virtualization engine to get the real image by dynamically generating an encryption key based on the calculated time. This real image is further used for threat and risk detection.

It is nearly impossible to crack this solution as time-dependent images are created using internal behavioural properties and dynamically generated properties, using dynamically generated code.

Similarly, it is nearly impossible to steal and reuse these images as they are relative and private to the user and their device. They have a very limited life and are one-time use.

This way the API payload is converted from static to dynamic. It becomes nearly impossible to exploit APIs with either stolen data or managed data.

**Since all images are dynamic and specific to the time on a user's phone, the only way to steal data is to physically have the user's phone and hack the phone at the exact moment in the past when the dynamic image was generated.**

**This is practically IMPOSSIBLE.**