Architecture-Driven Penetration Testing against an Identity Access Management (IAM) System

CAE Tech Talk

Thursday, September 201, 2018

Dr. Sam Chung, Professor/Director
Information Security Program
Center for Information Assurance Education
Technology Institute

Worked with
Sky Moon, MS
Expedia.com, Bellevue, WA 98004

Barbara Endicott-Popovsky, Ph.D.
Center for Info. Assurance & Cybersecurity
University of Washington, Bothell, WA

Published at ACM SIGITE/ RIIT 2016

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

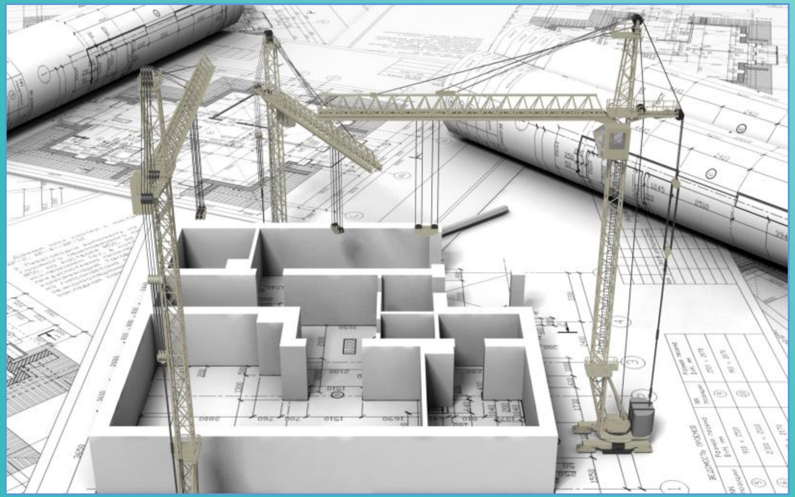
Agenda

- Motivation
- Problem Statement
- Background
- Previous Work
- Approach
- Architecture
- Architecture Modeling
- Vulnerabilities
- Demo
- Countermeasures
- Conclusion
- Future Work

Motivation – to make software attacks difficult

- Do not focus on blindly testing security functionality.

- Focus on **improving software architecture.**



Motivation – Why Architecture?

- **One half of all security problems come from design flaws**
- Performing a risk analysis **at the design level** is important.

(Verdon, D., and McGraw, G. 2004. Risk Analysis in Software Design. *IEEE Security & Privacy*, 2, 4 (Aug. 2004), 32-37.
IEEE Center for Secure Design, Avoiding the Top 10 Software Security Design Flaws, 2015)

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Problems

- Software security means the protection of software after it has been built & **deployed**.
- Challenges:
  - **How can we discover architectural design and abuse cases from a deployed system?**
  - **Based upon the architecture and abuse cases, how can we identify vulnerabilities and propose countermeasures for the deployed system?**

Case Study

- A telecommunication company in Washington had a plan to discover vulnerabilities of their Identity Assess Management (IAM) system before release.
- A question from a Vice President
  - How can vulnerabilities of the newly developed IAM system be identified and related vulnerabilities be mitigated?

WE'RE ALL ABOUT THE FINISH

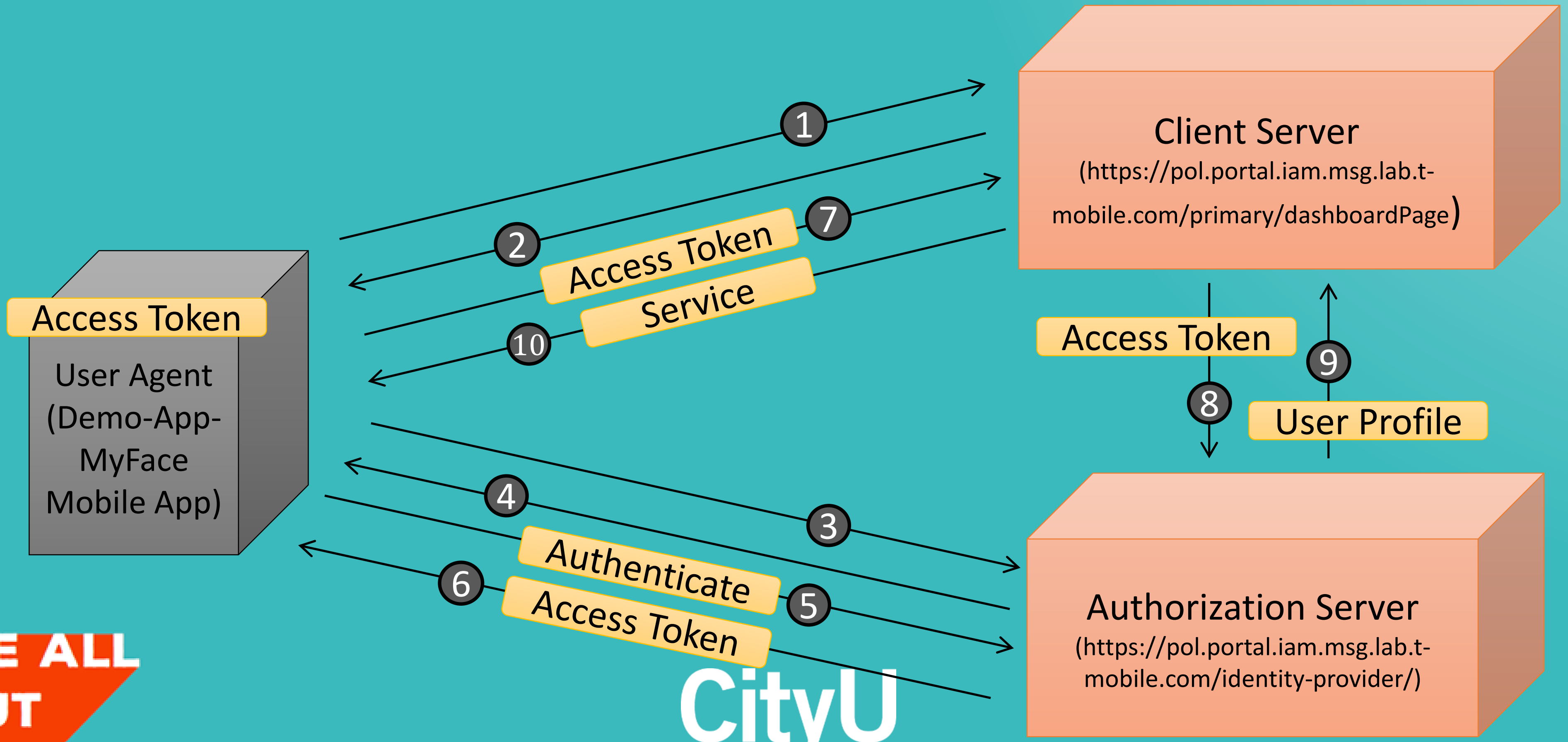
CityU of Seattle

Background

- Identity Access Management (IAM)
- Software Testing vs. Penetration Testing

Identity Access Management (IAM)

- Based on OAuth 2.0.



Client Server
(https://pol.portal.iam.msg.lab.t-mobile.com/primary/dashboardPage)
Authorization Server
(https://pol.portal.iam.msg.lab.t-mobile.com/identity-provider/)
Access Token
User Agent
(Demo-App-MyFace Mobile App)
1
2
7
Access Token
Service
10
Access Token
8
9
User Profile
4
3
Authenticate
5
6
Access Token

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Identity Access Management (IAM)

- *"A framework for business processes that facilitates the management of electronic identities."*
  (Rouse, M. 2015. Identity Access Management (IAM) System)

- IAM will be necessary in the future for managing data security of Bring-Your-Own-Device (BYOD) or Cloud Computing
  Cser A. and Maxim, M. 2016. IAM is the future for managing data security, (Mar. 2016), ComputerWeekly.com

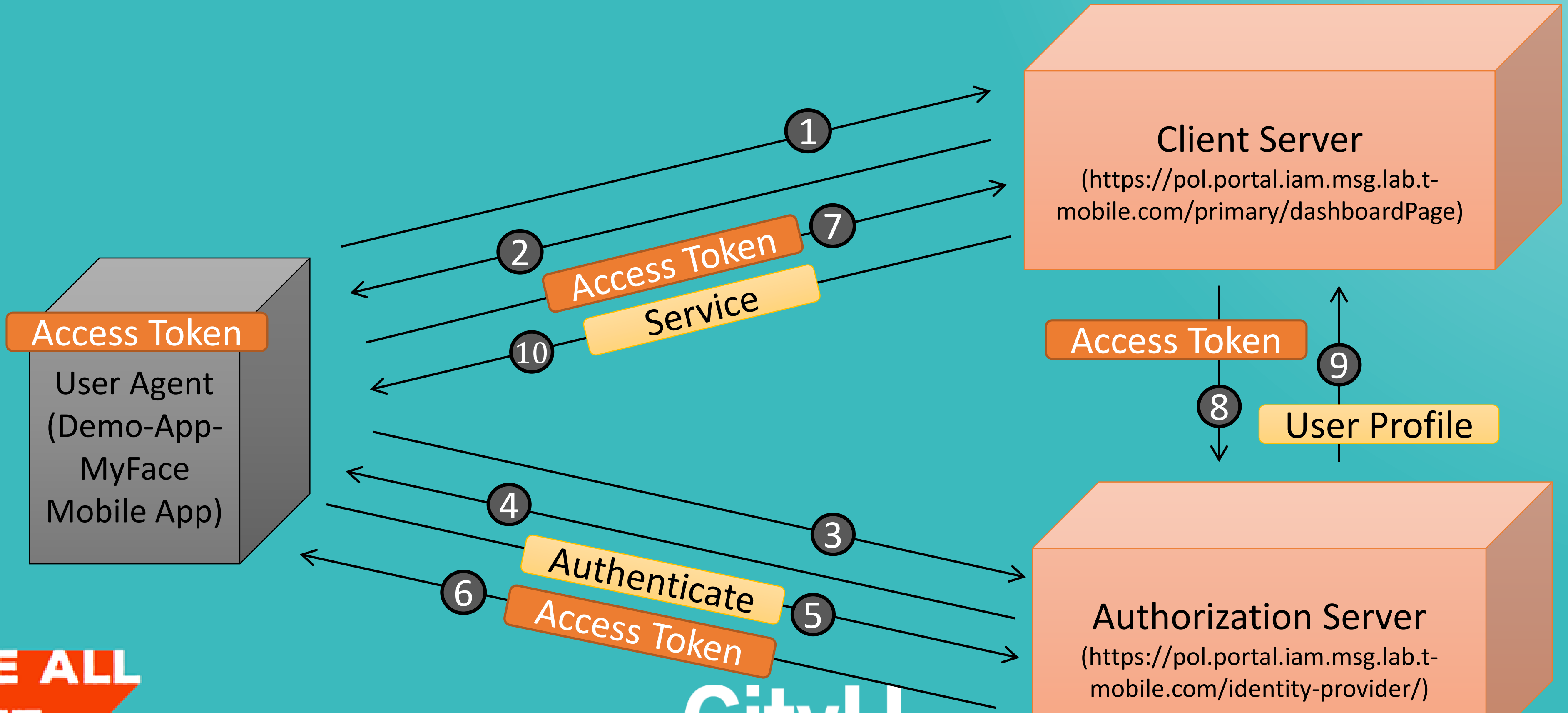
Software Testing vs. Penetration Testing

- Software Testing
  - A **normal** user’s perspective
  - **No approval** from the test requesters
  - Find the absence of a **specified behavior** of a given insecure legacy system.

- Penetration Testing
  - An **abnormal** user’s perspective
  - **Approval** from the test requesters
  - Find the absence of **an unspecified behavior** of a given insecure legacy system.

Approaches

- Our Target : “**Access Token**”



Client Server
(https://pol.portal.iam.msg.lab.t-mobile.com/primary/dashboardPage)
Access Token
User Agent
(Demo-App-MyFace Mobile App)
1
2
Access Token
7
Service
10
Access Token
9
8
User Profile
4
3
Authenticate
5
6
Access Token
Authorization Server
(https://pol.portal.iam.msg.lab.t-mobile.com/identity-provider/)

WE’RE ALL ABOUT THE FINISH

CityU of Seattle

Previous Work – Architectural Risk Analysis

- To discover software design flaws and abuse cases based upon those flaws in software security:
  - Arkin B., Stender, S., and McGraw, G. 2005. Software Penetration Testing, *IEEE Security & Privacy*, 3, 1, (Mar. 2005)
  - McGraw, Software Security. *IEEE Security & Privacy*, 2, 2 (Apr. 2004), 80-83.
  - Potter, B., and McGraw, G. 2004. Software Security Testing, *IEEE Security & Privacy*, 2, 5 (Oct. 2004), 81-85.
  - Thomson, H. H. 2005. Application Penetration Testing, *IEEE Security & Privacy*, 3, 1 (Feb. 2005), 66-69

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Previous Work

- Although the importance of architectural risk analysis has been proposed a decade ago, those articles found focus on using architecture for risk analysis, as opposed to **discovering the architecture of a given insecure legacy system**.
- Borrow the approach from software reengineering.

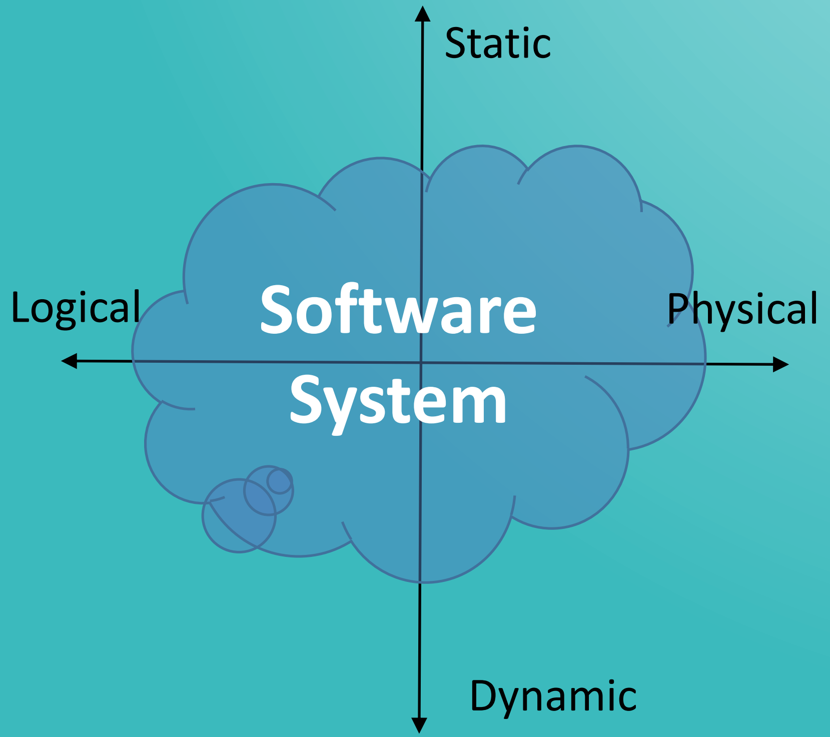
Approach - Architecture-Driven, Penetration Testing Methodology

- An reengineer an insecure legacy system to a secure target system
  - by discovering use cases for normal users and **abuse cases for hackers**
  - through **a reverse engineering process** which identifies vulnerabilities based upon the abuse cases, and
  - proposes countermeasures that will be used through **a forward engineering process**.

Software Architecture

- **The architecture of a given insecure legacy system will be the main information for penetration testi**ng.

- Through the reverse engineering process, the architecture of the legacy system is re-documented into **a visual model that explains physical/logical and static/dynamic properties of the system**.



WE'RE ALL ABOUT THE FINISH

CityU of Seattle

4+1 View Model of Architecture

(Kruchten, IEEE Software, 1995)

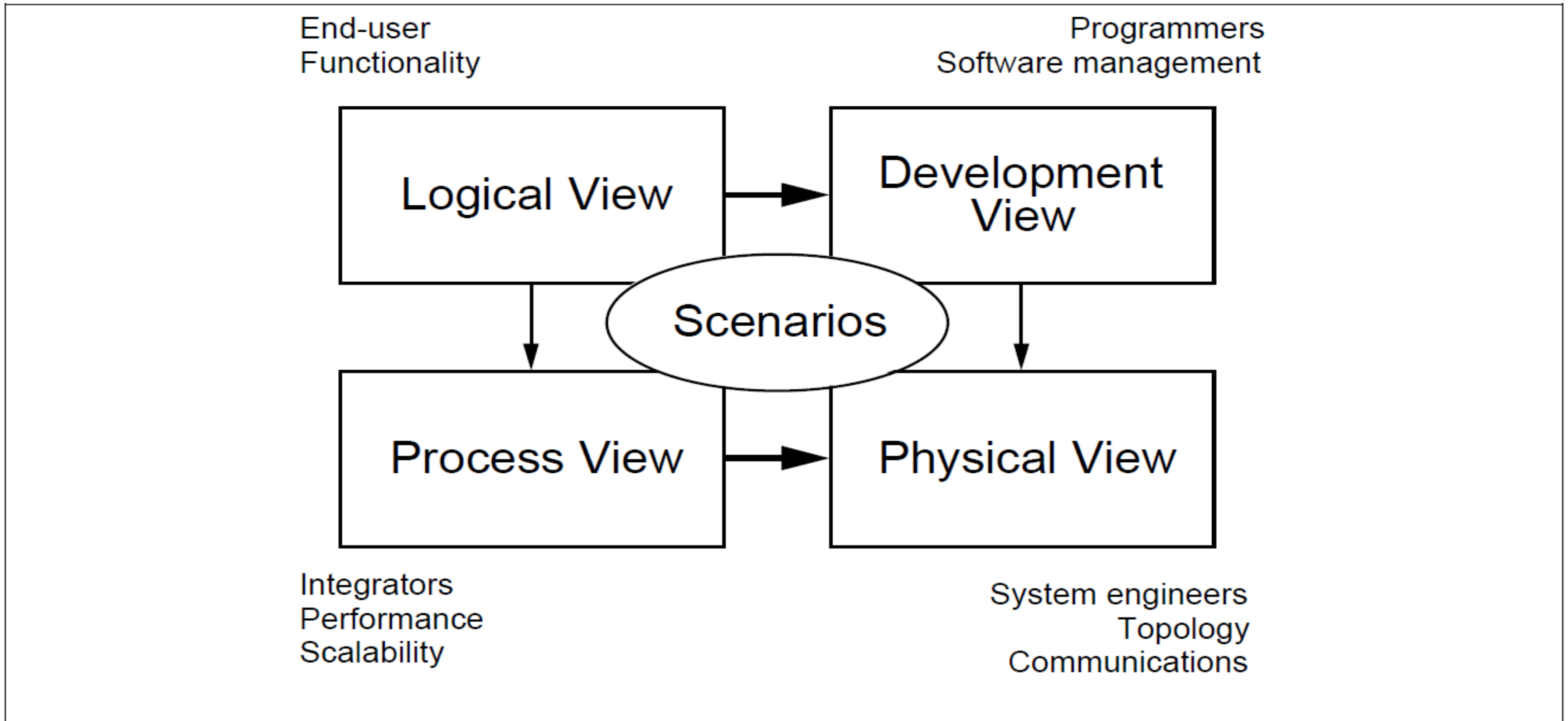


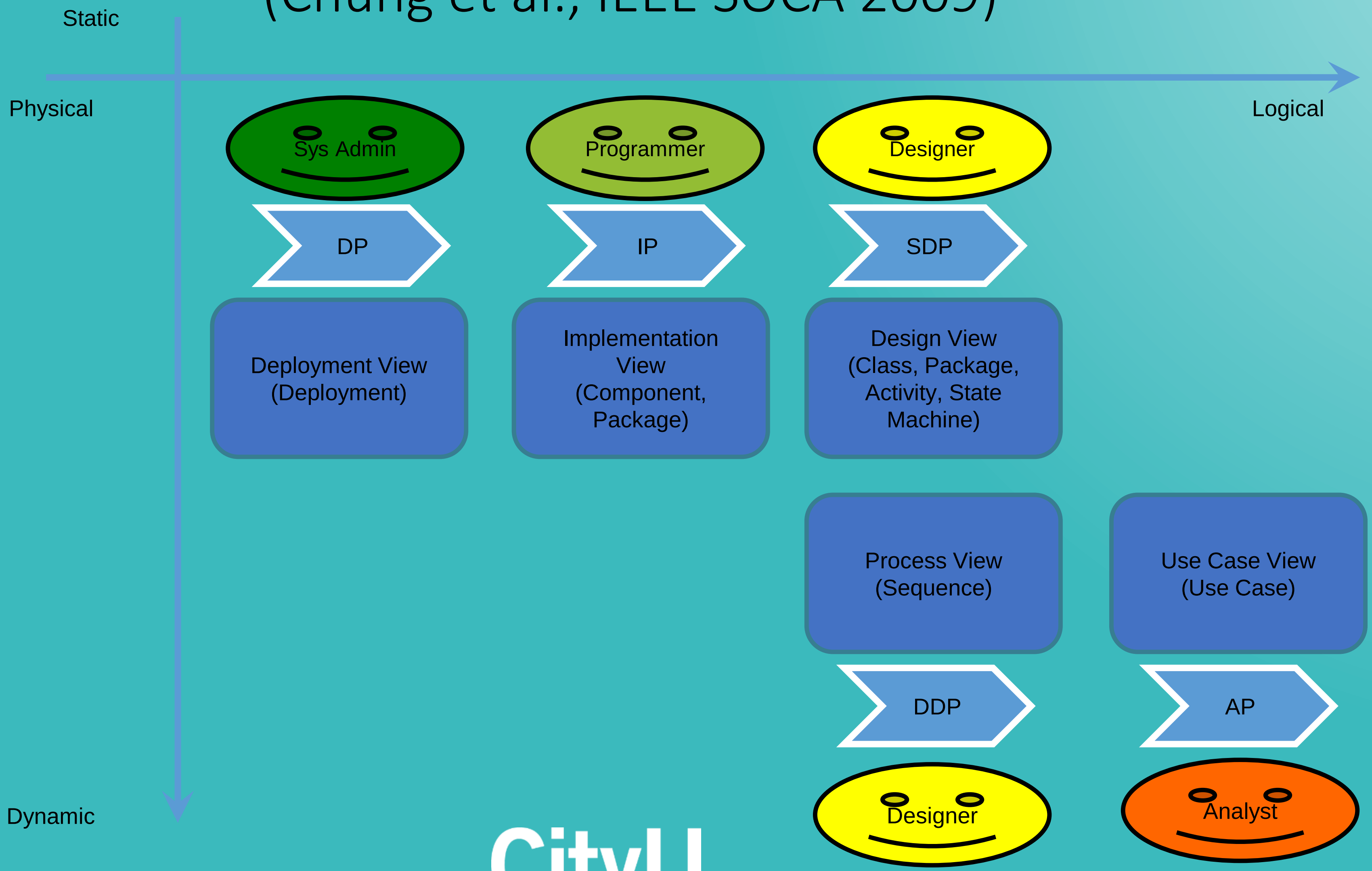
Figure 1 — The “4+1” view model

WE’RE ALL ABOUT THE FINISH

CityU of Seattle

5W1H Re-Doc

(Chung et al., IEEE SOCA 2009)



WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Spoofing Identity Attack

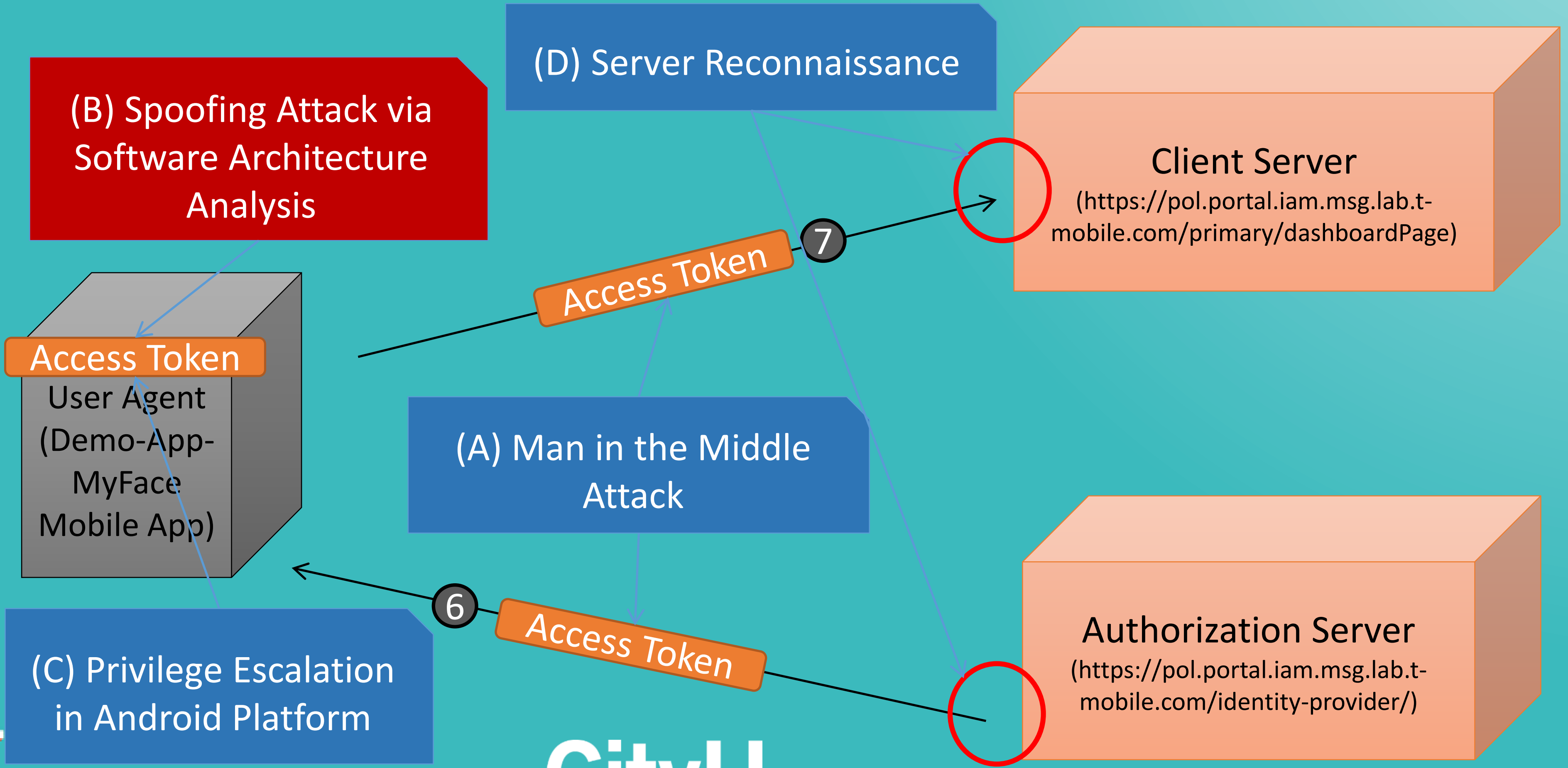
- Is the spoofing identity attack possible?
  - Conditionally, Yes.

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Approaches

- Detailed strategies

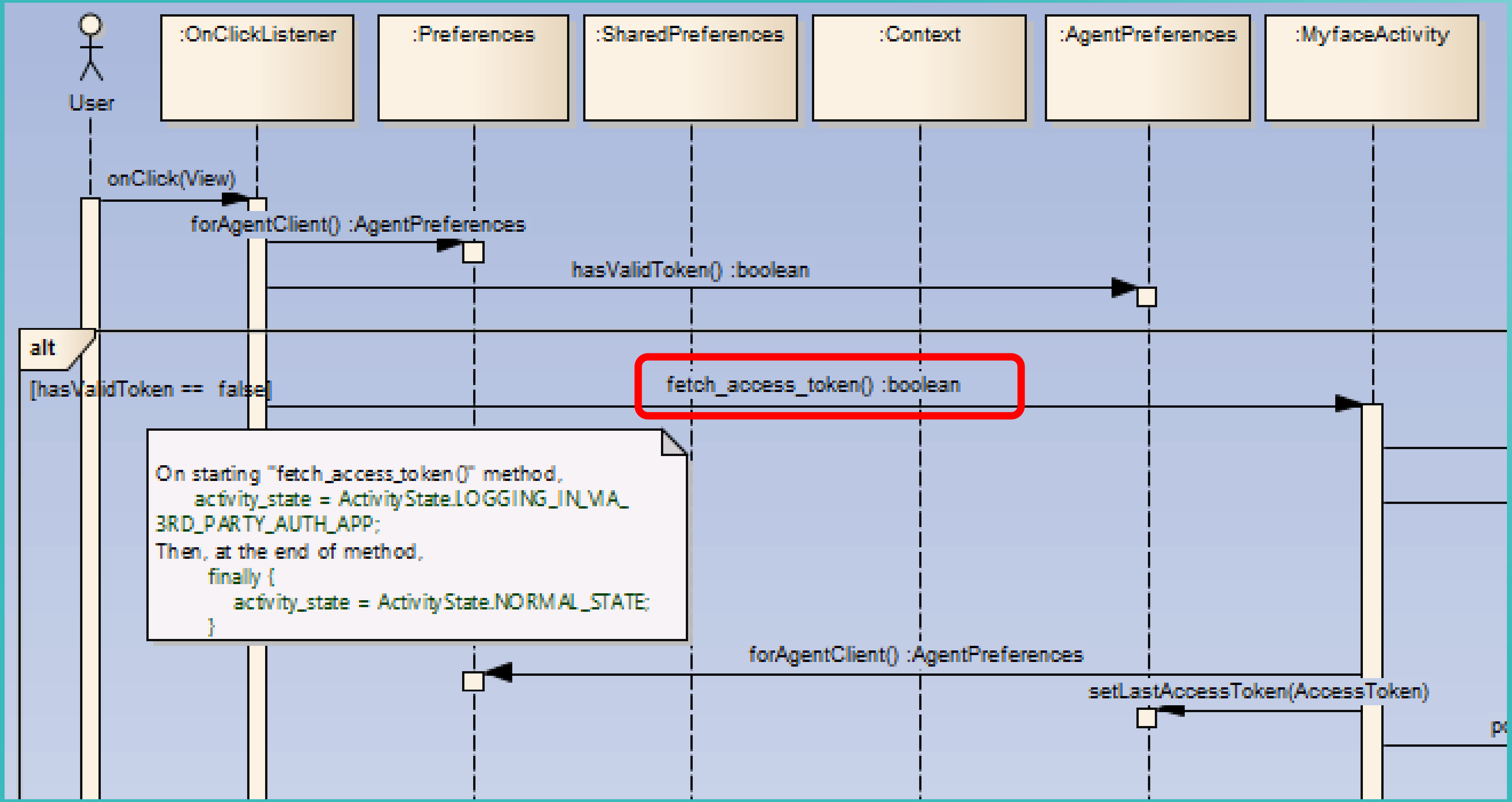


(D) Server Reconnaissance
(B) Spoofing Attack via Software Architecture Analysis
Client Server
(https://pol.portal.iam.msg.lab.t-mobile.com/primary/dashboardPage)
7
Access Token
Access Token
User Agent (Demo-App-MyFace Mobile App)
(A) Man in the Middle Attack
6
Access Token
Authorization Server
(https://pol.portal.iam.msg.lab.t-mobile.com/identity-provider/)
(C) Privilege Escalation in Android Platform

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

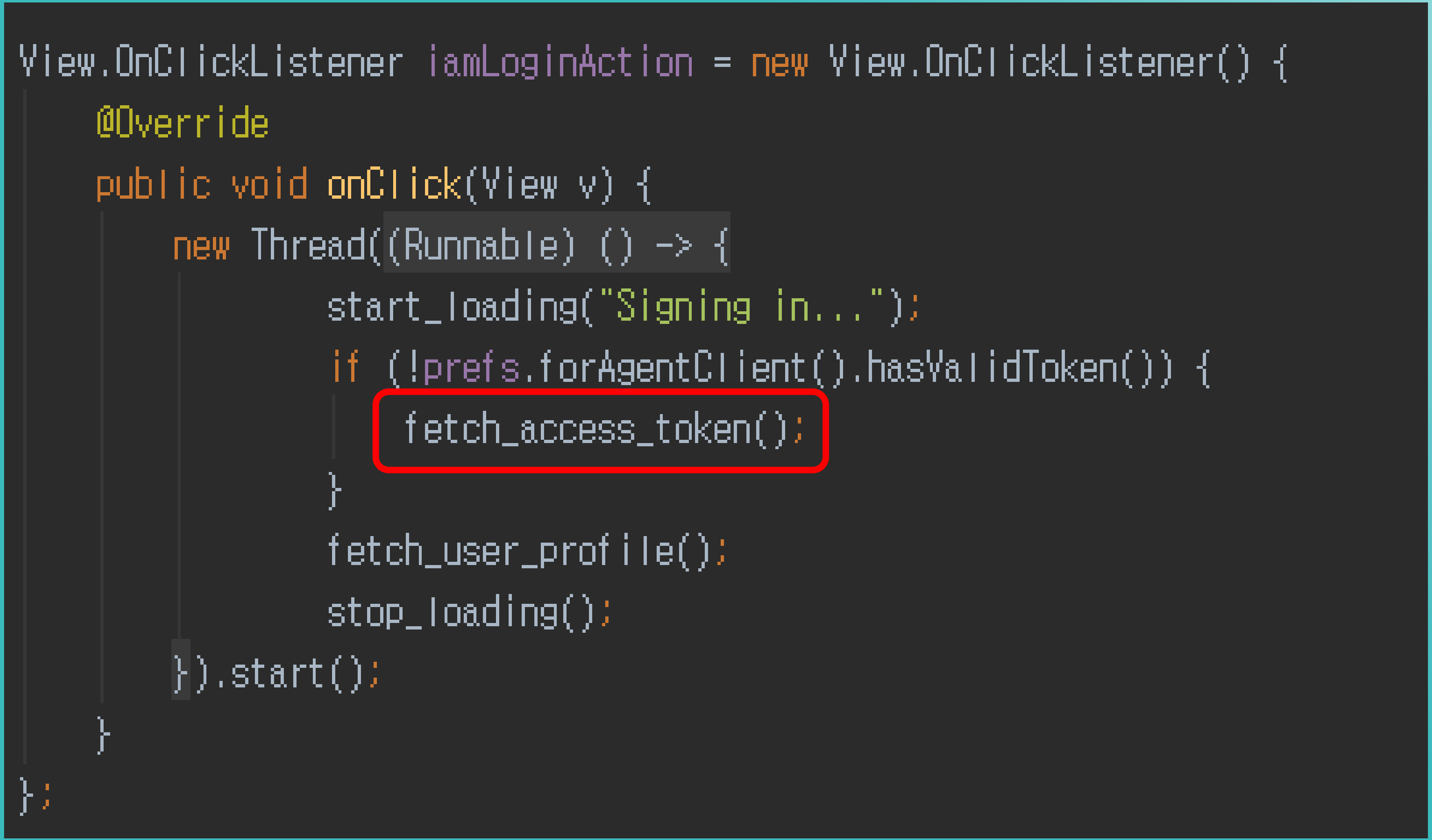
Software Architecture Analysis of Demo-App-MyFace: Fetching an Access Token



```
On starting "fetch_access_token()" method,
     activity_state = ActivityState.LOGGING_IN_VIA_
3RD_PARTY_AUTH_APP;
Then, at the end of method,
       finally {
           activity_state = ActivityState.NORMAL_STATE;
       }
```

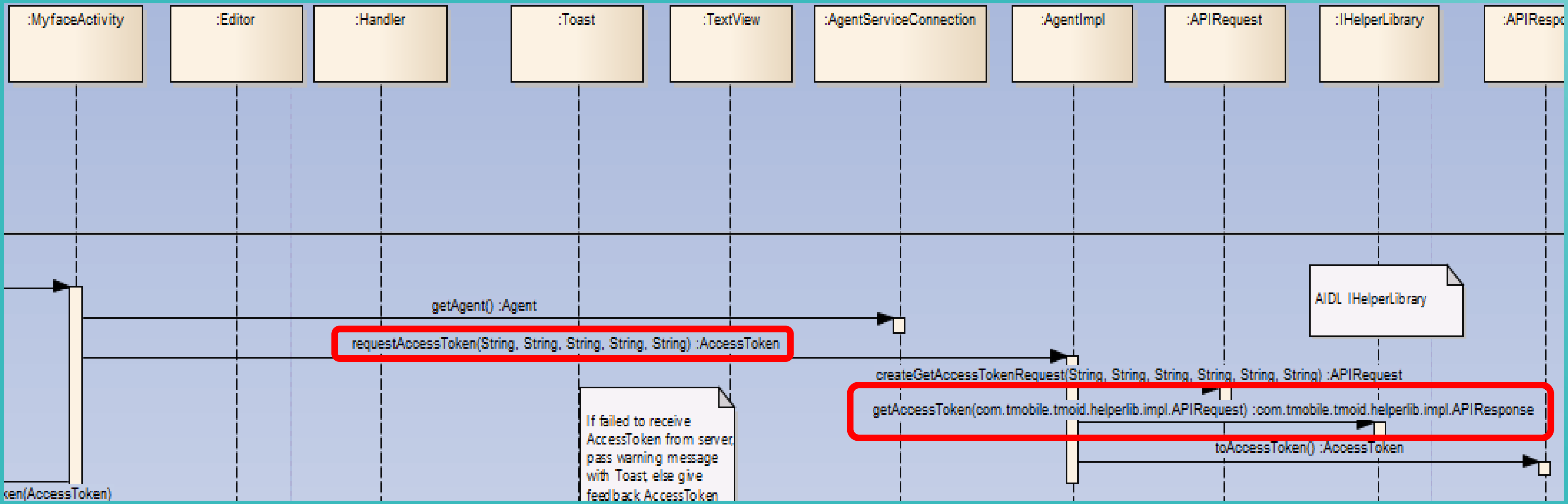
WE'RE ALL ABOUT THE FINISH

CityU
of Seattle

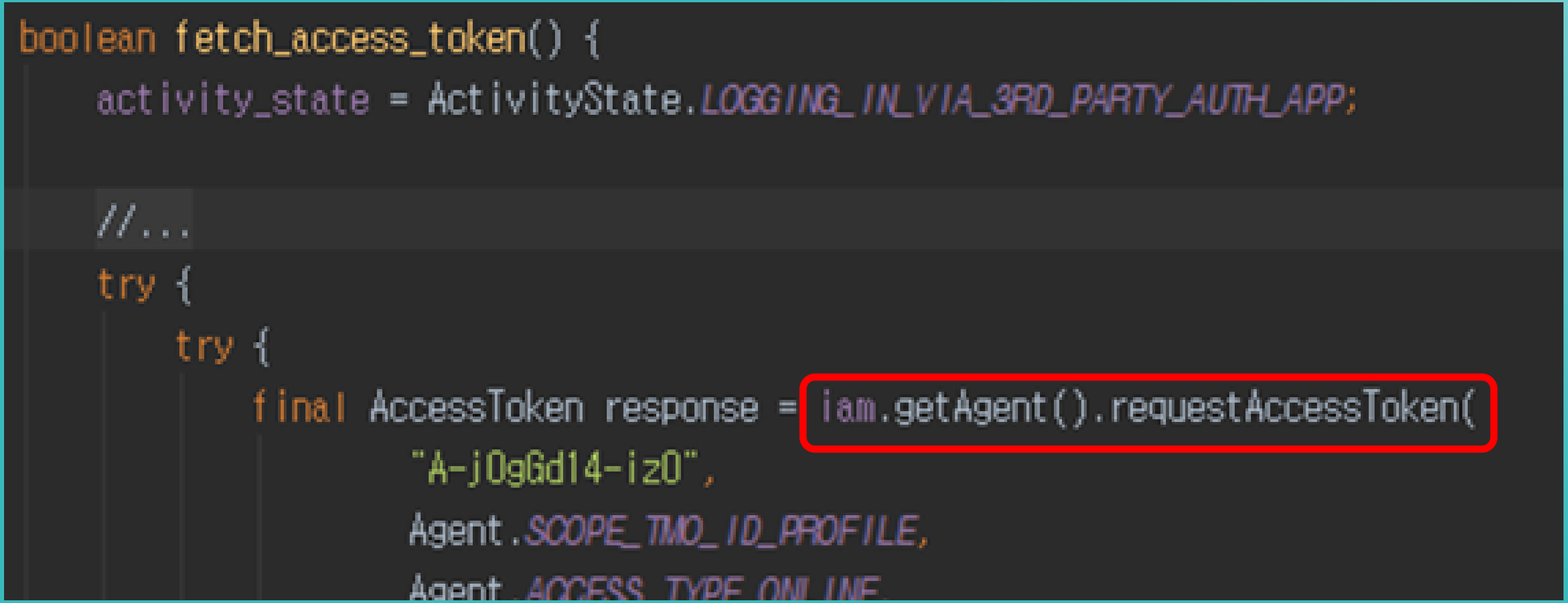


```
View.OnClickListener iamLoginAction = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new Thread((Runnable) () -> {
                start_loading("Signing in...");
                if (!prefs.forAgentClient().hasValidToken()) {
                    fetch_access_token();
                }
                fetch_user_profile();
                stop_loading();
        }).start();
    }
};
```

Software Architecture Analysis of Demo-App-MyFace: Fetch an Access Token (Continued)



:MyfaceActivity
:Editor
:Handler
:Toast
:TextView
:AgentServiceConnection
:AgentImpl
:APIRequest
:IHelperLibrary
:APIResp
getAgent() :Agent
requestAccessToken(String, String, String, String, String) :AccessToken
createGetAccessTokenRequest(String, String, String, String, String, String) :APIRequest
getAccessToken(com.tmobile.tmoid.helperlib.impl.APIRequest) :com.tmobile.tmoid.helperlib.impl.APIResponse
toAccessToken() :AccessToken
AIDL IHelperLibrary
If failed to receive AccessToken from server, pass warning message with Toast else give feedback AccessToken
ken(AccessToken)

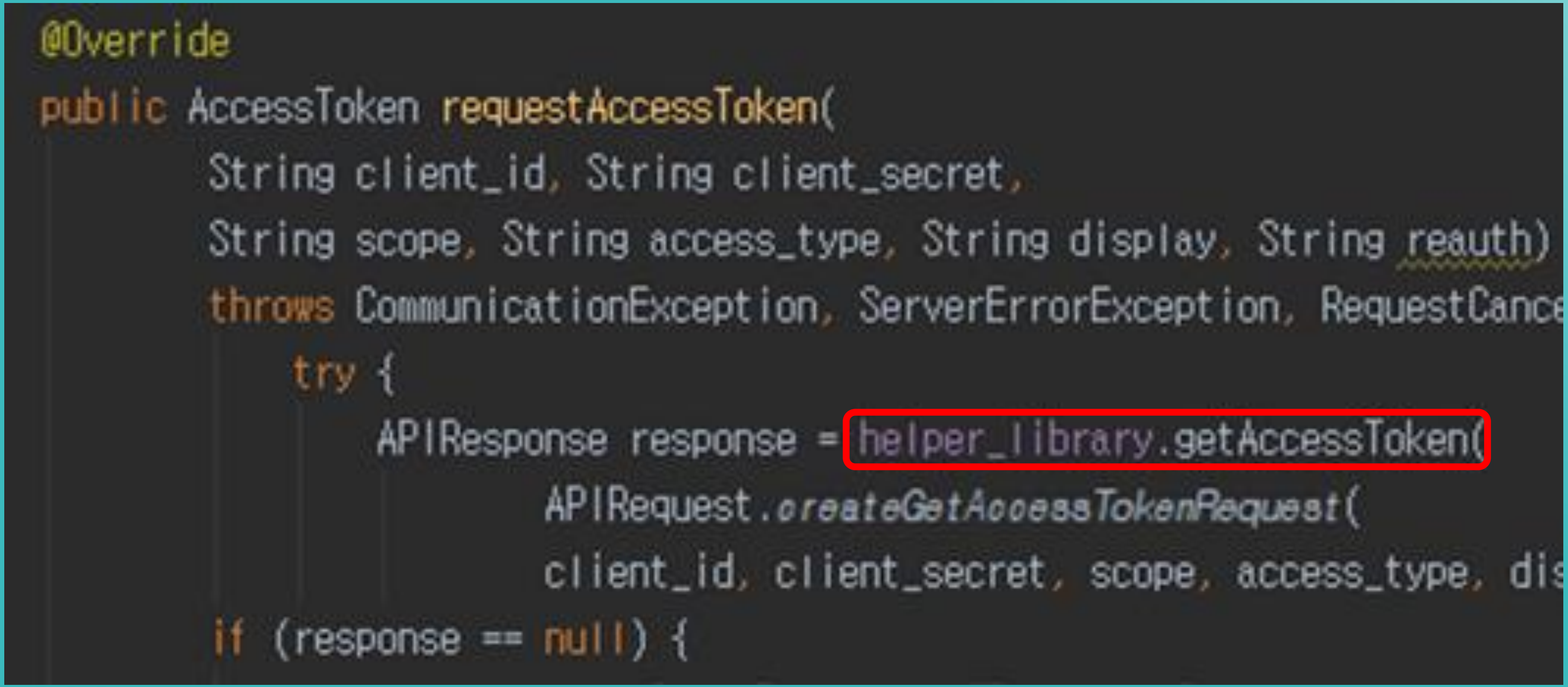


```
boolean fetch_access_token() {
    activity_state = ActivityState.LOGGING_IN_VIA_3RD_PARTY_AUTH_APP;

    //...
    try {
        try {
            final AccessToken response = iam.getAgent().requestAccessToken(
                    "A-j0gGd14-iz0",
                    Agent.SCOPE_TMO_ID_PROFILE,
                    Agent.ACCESS_TYPE_ONLINE,
```

WE'RE ALL ABOUT THE FINISH

CityU of Seattle



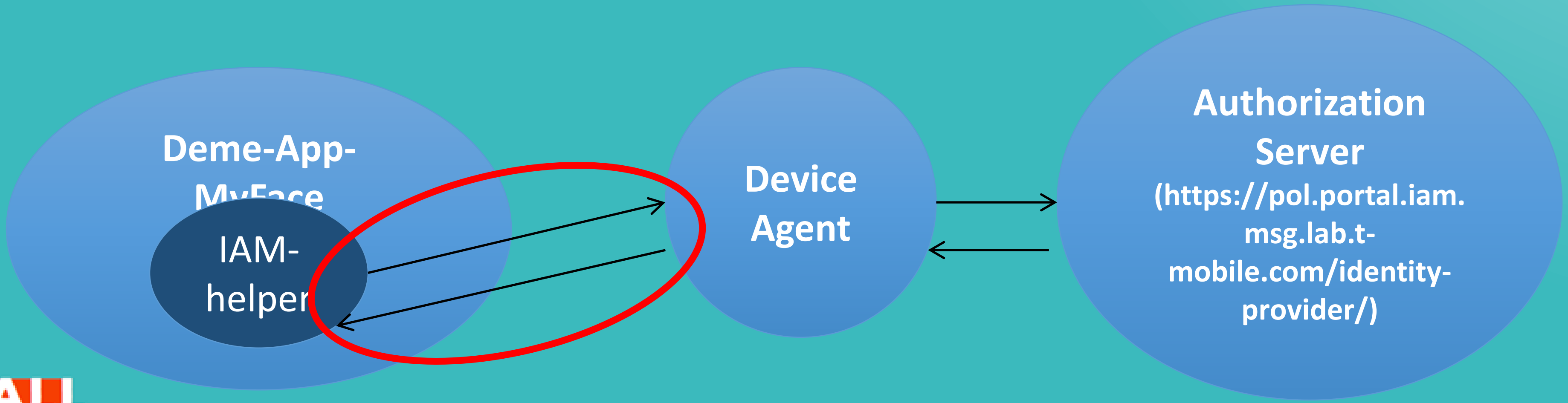
```
@Override
public AccessToken requestAccessToken(
        String client_id, String client_secret,
        String scope, String access_type, String display, String reauth)
        throws CommunicationException, ServerErrorException, RequestCance
            try {
                APIResponse response = helper_library.getAccessToken(
                        APIRequest.createGetAccessTokenRequest(
                        client_id, client_secret, scope, access_type, dis
        if (response == null) {
```

WE'RE ALL
ABOUT
THE FINISH

CityU
of Seattle

Android Interface Definition Language (AIDL)

- Used for data exchange between iam-helper used in Demo-App-MyFace and Device Agent

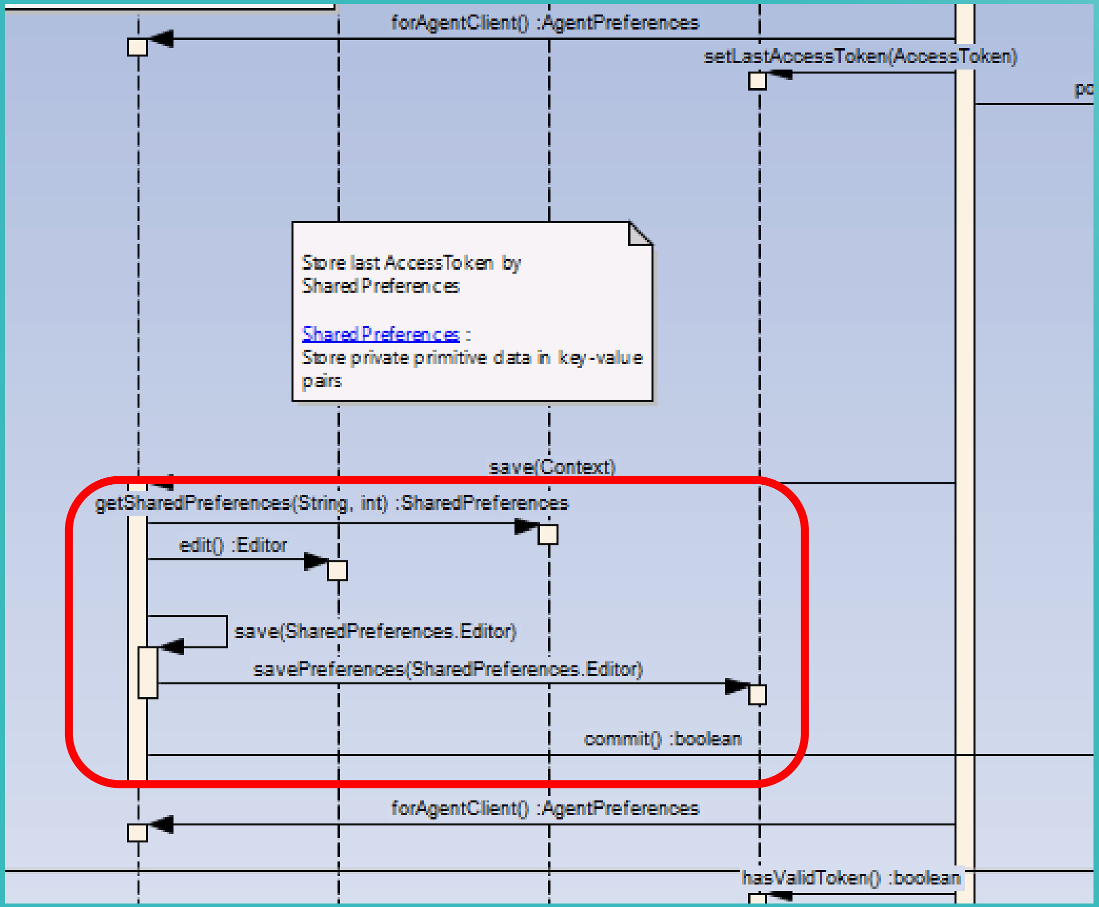


Deme-App-MyFace
IAM-helper
Device Agent
Authorization Server
(https://pol.portal.iam.msg.lab.t-mobile.com/identity-provider/)

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Software Architecture Analysis of Demo-App-MyFace: Storing an Access Token with ‘SharedPreferences’ into a XML File


forAgentClient() :AgentPreferences
setLastAccessToken(AccessToken)
po
save(Context)
getSharedPreferences(String, int) :SharedPreferences
edit() :Editor
save(SharedPreferences.Editor)
savePreferences(SharedPreferences.Editor)
commit() :boolean
forAgentClient() :AgentPreferences
hasValidToken() :boolean

Store last AccessToken by SharedPreferences

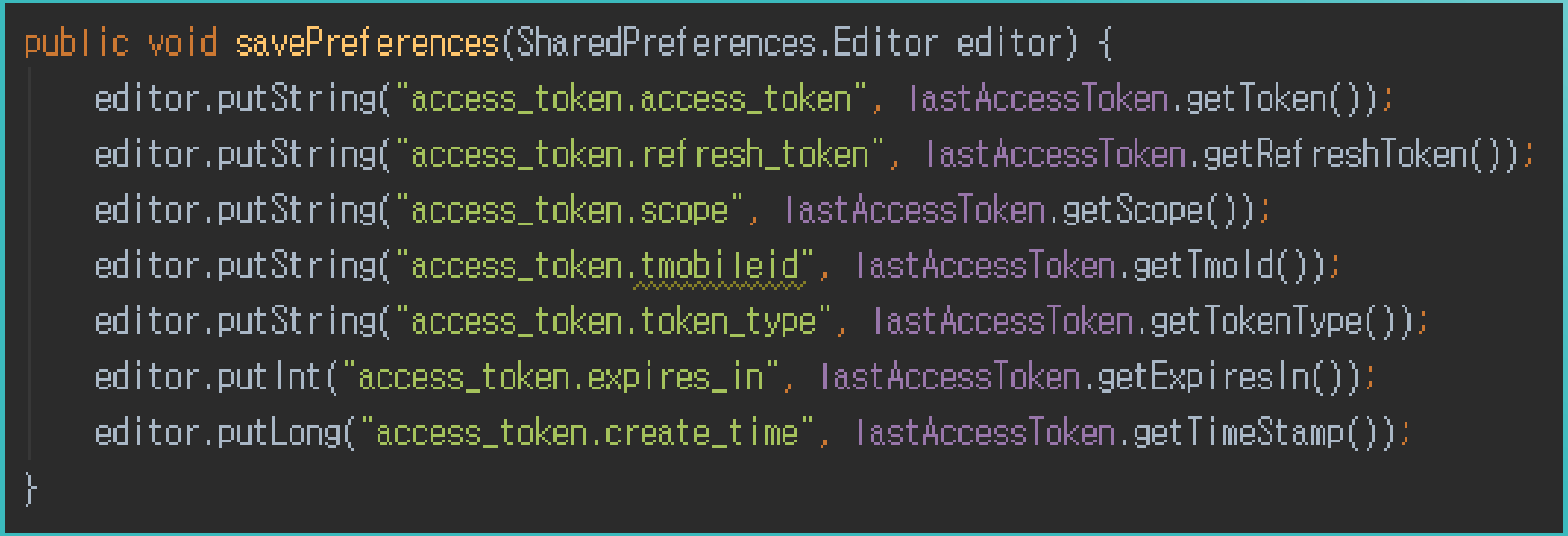
SharedPreferences :
Store private primitive data in key-value pairs

Two Possibly Vulnerable Points in the Demo-App-MyFace & iam-helper Library

- AIDL connection between the Demo-App-MyFace and Device Agent

- The access token stored by the SharedPreferences (It is unsecure).

The Access Token is Stored Using the SharedPreferences



```java
public void savePreferences(SharedPreferences.Editor editor) {
    editor.putString("access_token.access_token", lastAccessToken.getToken());
    editor.putString("access_token.refresh_token", lastAccessToken.getRefreshToken());
    editor.putString("access_token.scope", lastAccessToken.getScope());
    editor.putString("access_token.tmobileid", lastAccessToken.getTmoId());
    editor.putString("access_token.token_type", lastAccessToken.getTokenType());
    editor.putInt("access_token.expires_in", lastAccessToken.getExpiresIn());
    editor.putLong("access_token.create_time", lastAccessToken.getTimeStamp());
}
```

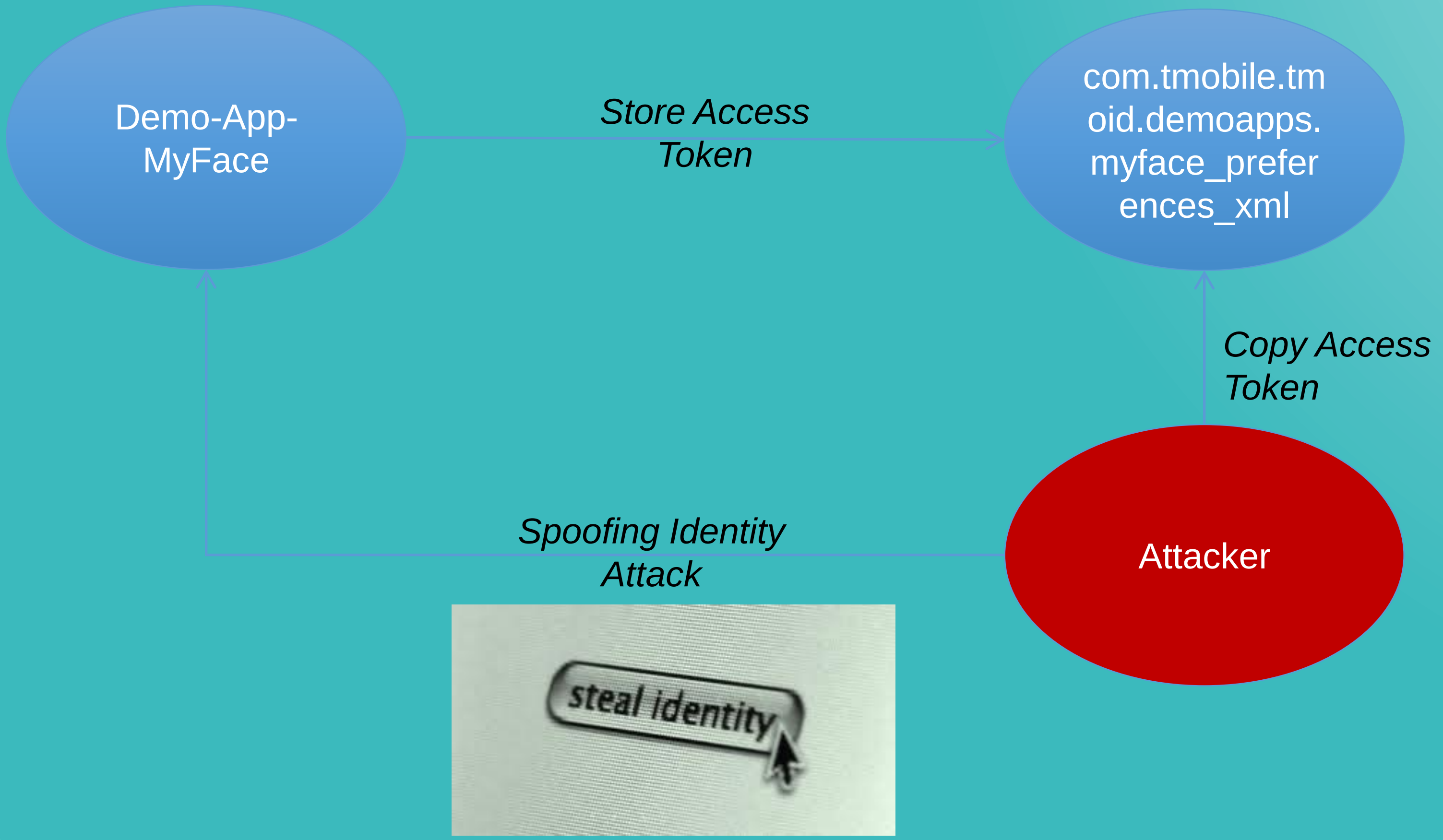
Shared Preferences

- Store private primitive data in key-value pairs into a XML file.

{Package name}_preferences.xml

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
   <string name=“KEY">VALUE</string>
</map>
```

Spoofing Identity Attack



Demo

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Countermeasures

- We successfully obtained a user profile from the resource server using the access token extracted from the Android file system.
- For each identified vulnerability for Android app and server endpoints, we recommend two reliable countermeasures, with references to RFC 6819, for **the Android app** and **the server endpoint vulnerabilities**, respectively (OAuth, 2016).

Countermeasures

- The following countermeasures are proposed for the Android app vulnerabilities:
  - Do not log the access token retrieval part (RFC6819 Section 4.6.7). Accidently, developers of the ‘iam-helper’ library did not remove the logs for the access token retrieval.
  - Use Authorization headers or POST parameters instead of URI request parameters (RFC 6819 Section 5.4.1) - “Authorization headers are recognized and specially treated by HTTP proxies and servers. Thus, the usage of such headers for sending access tokens to resource servers reduces the likelihood of leakage or unintended storage of authenticated requests in general, and especially Authorization headers.”
  - …

Countermeasures (Continued)

- The following countermeasures are proposed for the Android app vulnerabilities:
  - Keep the access token in transient memory and limit grants (RFC6819 Section 5.1.6). The access token should not be stored in a physical file system. There may be a way to get data even from transient memory, but it would be much more difficult.
  - Keep the access token in private memory or apply the same protection means as for refresh tokens (RFC6819 Section 5.2.2). We also need to store the refresh token in private memory for the refresh token. Do not store it in a physical file system.
  - Limit the access token's scope (RFC6819 Section 5.1.5.1). It is better to limit the privilege of the access token, if you implemented the privilege mechanism.
  - Keep the access token's lifetime short (RFC6819 Section 5.1.5.3.) The shorter the lifetime, the more secure your system. Currently the lifetime is one hour.

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Countermeasures (Continued)

- A countermeasure proposed for the server endpoints vulnerability follows:
  - Insert a blocking mechanism (i.e., blocking a resource request from the same IP address, if it fails more than 3 times within a time interval) to prevent a brute-force attack.

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

Conclusions

- In order to discover architectural design and abuse cases from a deployed insecure legacy system, we borrowed **ideas from software reengineering**: we consider a given system as a legacy system that may have security vulnerabilities, reverse engineer the given legacy system to identify possible vulnerabilities, and then propose countermeasures for a target system that won't have those vulnerabilities.
- We apply a reverse engineering methodology called **5W1H Re-Doc** to a given legacy system and discover the system architecture from the hacker's view.

Discussion

- Spoofing Identity attack is possible if and only if an attacker has a root permission.
- Do not store the access token into a shared human readable XML file.
- Question:
  Why are you storing the access token in ‘Demo-App-MyFace’ into a shared XML file?

WE’RE ALL ABOUT THE FINISH

CityU of Seattle

Future Work

- A promising future for architecture-driven penetration testing
  - To help a security engineer identify vulnerabilities from **nothing (black-box penetration testing)** to **architecture (white-box penetration testing**)
  - To prepare for countermeasures against identified vulnerabilities by considering **both physical and cyber properties with multiple and hierarchical architectural views**.

WE'RE ALL ABOUT THE FINISH

CityU of Seattle

WE'RE ALL
ABOUT
THE FINISH

CityU
of Seattle