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
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.

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?
Background

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

- Based on OAuth 2.0.

**User Agent** (Demo-App - MyFace Mobile App)

**Access Token**

**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 Profile**

**Authenticate**

**Access Token**

**Service**

**Access Token**
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
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”
Previous Work – Architectural Risk Analysis

• To discover software design flaws and abuse cases based upon those flaws in software security:
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 testing.

• 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.
4+1 View Model of Architecture
(Kruchten, IEEE Software, 1995)

Figure 1 — The “4+1” view model
5W1H Re-Doc
(Chung et al., IEEE SOCA 2009)
Spoofing Identity Attack

• Is the spoofing identity attack possible?
  • Conditionally, Yes.
Approaches

- Detailed strategies

(A) Man in the Middle Attack

(B) Spoofing Attack via Software Architecture Analysis

(C) Privilege Escalation in Android Platform

(D) Server Reconnaissance

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

On starting "fetch_access_token" method,
activity_state = ActivityState.LOGINING_IN VIA 3RD PARTY AUTH APP.
Then, at the end of method,
finally {
    activity_state = ActivityState.NORMAL_STATE;
}
View.OnClickListener iamLoginAction = new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        new Thread(() -> {
            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)
boolean fetch_access_token() {
    activity_state = ActivityState.LOGGING_IN_VIA_3RD_PARTY_AUTH_APP;

    // ...
    try {
        try {
            final AccessToken response = iam.getAgent().requestAccessToken(
                "A-jOgGd14-iz0",
                Agent.SCOPE_TMO_ID_PROFILE,
                Agent.ACCESS_TYPE_ONLINE
            );
        }
    }
}
```java
@override
public AccessToken requestAccessToken(
    String clientId, String clientSecret,
    String scope, String accessTokenType, String display, String reauth)
throws CommunicationException, ServerErrorException, RequestCanceledException
{
    try {
        APIResponse response = helper_library.getAccessToken(
            APIRequest.createGetAccessTokenRequest(
                clientId, clientSecret, scope, accessTokenType, display)
        );
        if (response == null) {
```
Android Interface Definition Language (AIDL)

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

Diagram:
- Deme-App-MyFace
- IAM-helper
- Device Agent
- Authorization Server (https://pol.portal.iam.msg.lab.t-mobile.com/identity-provider/)
Software Architecture Analysis of Demo-App-MyFace: Storing an Access Token with ‘SharedPreferences’ into a XML File
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.getTmobileId());
    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.

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

Demo-App-MyFace

Store Access Token

com.tmobile.tmoid.demoapps.mylife_preferences_xml

Copy Access Token

Spoofing Identity Attack

Attacker

CityU of Seattle

WE'RE ALL ABOUT THE FINISH
Demo
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). Accidentally, 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.
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.
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?
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.