iOS APPLICATION PENETRATION TESTING

Report for:	
Date:	

This document contains confidential information about IT systems and network infrastructure of the customer, as well as information about potential vulnerabilities and methods of their exploitation. This confidential information is for internal use by the customer only and shall not be disclosed to third parties.

Table of Contents

Introduction

We thank _____ for giving us the opportunity to conduct Security Assessment of their mobile application and its backend API. This document outlines our methodology, limitations and results of the security assessment.

Executive Summary

Hackcontrol (Consultant) was contracted by _____ (Customer) to conduct the penetration testing of their mobile application.

This report presents the findings of the penetration testing of CLIENT`s mobile application conducted between February 04th, 2018 – February 22nd, 2018.

The main subject of the penetration testing is _____`s mobile systems & API.

Application Security Assessment has the following objectives:

- identify technical and functional vulnerabilities;
- estimate their severity level (ease of use, impact on information systems, etc);
- modelling the "most likely" attack vector against the Customer's Information System;
- proof of concept and exploitation of vulnerabilities;
- draw up a prioritized list of recommendations to address identified weaknesses.

According to our research, the mobile application is of **high security rating** for Customer and Backend systems; Several high-level vulnerabilities have been detected, however it requires a considerable amount of time and efforts to exploit them.

Three (3) High vulnerabilities of sensitive info logging and bypass root and developer mode checks were diagnosed during the security assessment. Also, three (3) Medium and a number of Low and Informative vulnerabilities and errors were identified.

Team

Role	Name	EMAIL
Project Manager	John Doe (CEH, ISO27001 LA)	info@hackcontrol.org
Penetration Testing Engineer	John Doe (OSCP, eWPT, eCPPT)	engineer@hackcontrol.org

Scope of Security Assessment

The following list of systems was in the scope of the Security Assessment.

#	Name	Description
1	__	iOS

Security Assessment start and end dates were coordinated by email according to the following table.

Methodology

The testing methodology is based on generally accepted industry-wide approaches to perform penetration testing for mobile applications – Mobile Security Testing Guide (MSTG);

Application-level penetration tests include, at a minimum, checking for the following types of vulnerabilities:

- lack of binary protections;
- insecure data storage;
- unintended data leakage;
- client-side injection;
- weak encryption;
- implicit trust of all certificates;
- execution of activities using root;
- private key exposure;
- exposure of database parameters and SQL queries;
- insecure random number generator;

Severity Definition

The level of criticality of each risk is determined based on the potential impact of loss from successful exploitation as well as ease of exploitation, existence of exploits in public access and other factors.

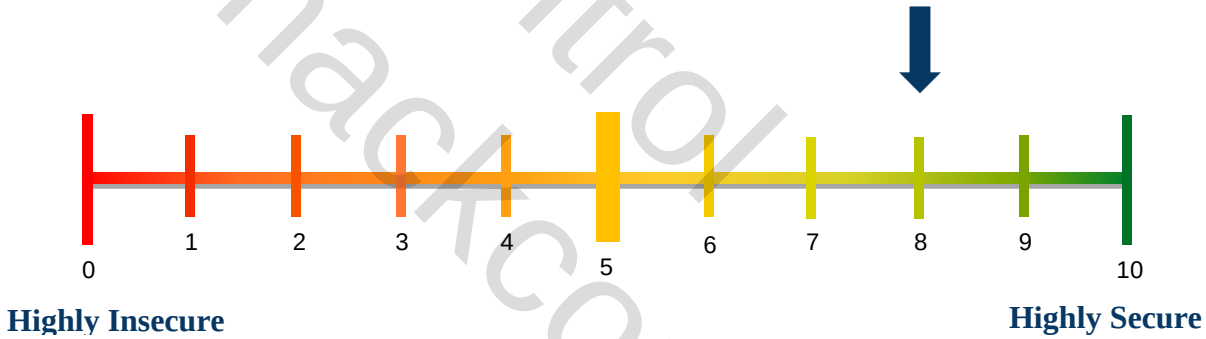
Severity	Description
High	High-level vulnerabilities are easy in exploitation and may provide an attacker with full control of the affected systems, also may lead to significant data loss or downtime. There are exploits or PoC available in public access.
Medium	Medium-level vulnerabilities are much harder to exploit and may not provide the same access to affected systems. No exploits or PoCs available in public access. Exploitation provides only very limited access.
Low	Low-level vulnerabilities provide an attacker with information that may assist them in conducting subsequent attacks against target information systems or against other information systems, which belong to an organization. Exploitation is extremely difficult, or impact is minimal.
Info	These vulnerabilities are informational and can be ignored.

Summary of Findings

According to the following in-depth testing of the environment, CLIENT iOS application requires some improvements.

Value	Numbers of risks
High	2
Medium	2
Low	2
Info	1

Based on our understanding of the iOS application, as well as the nature of the vulnerabilities discovered, their exploitability, and the potential impact we have assessed the level of risk for your organization to be **Low**.



No major design flaws were identified. No data manipulation or corruption were discovered, however some vulnerabilities against application availability and users' security are the point of concern. The vulnerabilities identified were the following: "User's credential stores locally and not encrypted in application's sandbox", "Requests and responses stores insecure in Cache.db", "Weak cryptography" and others.

Key Findings

User’s credential stores locally and not encrypted in application’s sandbox

#1	Description	Type: Real
Local database from /var/mobile/Containers/Data/Application/DC6488D9-C54A-4FE8-87DB-49764E92938C/Library/Caches/com.CLIENT.ff stores user’s credential		
Evidences		
Steps to reproduce: - Sign up/Log in to the application - Connect to the device with ssh - Navigate to application's sandbox - Open Cache.db with any SQLite viewer, from /Library/Caches/com.company.exchange/		

```
b0  62 39 37 64 38 36 66 30 63 34 36 32 31 66 61 32   b97d86f0c4621fa2
c0  62 30 65 31 37 63 61 62 64 38 63 26 70 68 6f 6e   b0e17cabd8c&phon
d0  65 43 6f 64 65 3d 36 38 34 38 38 35 26 72 65 67   eCode=684885&reg
e0  4e 61 6d 65 3d 39 35 31 37 4 0 30 35 31 33 26 72   Name=951740513&r
f0  65 67 54 79 70 65 3d 30 00 08 00 0d 00 15 00 1b   egType=0........
00  00 1d 00 36 00 37 00 3c 00 4f 00 5c 00 5e 00 8d   ...6.7.<.O.\.^.
10  00 96 00 98 00 ba 00 bb 00 bd 00 be 00 bf 00 c8   ..... .°.».¼.¾.¿.È
20  00 d1 00 d3 00 d8 00 eb 00 f2 00 ff 01 11 01 1e   .Ñ.Ó.Ø.ë.ò.ÿ....
30  01 2a 01 35 01 44 01 56 01 62 01 66 01 99 01 d3   .*.5.D.V.b.f. .Ó
40  01 da 01 de 02 49 02 4c 02 69 02 90 02 92 00 00   .Ú.Þ.I.L.i. . ..
50  00 00 00 00 02 01 00 00 00 00 00 00 00 2b 00 00   .............+..
```

Recommendations	Application shouldn`t stores locally user`s credentials

Requests and responses stores insecure in cache.db

#2	Description	Type: Real

Local database from /var/mobile/Containers/Data/Application/DC6488D9-C54A-4FE8-87DB-49764E92938C/Library/Caches/com.company.exchange stores all requests and responses with data.

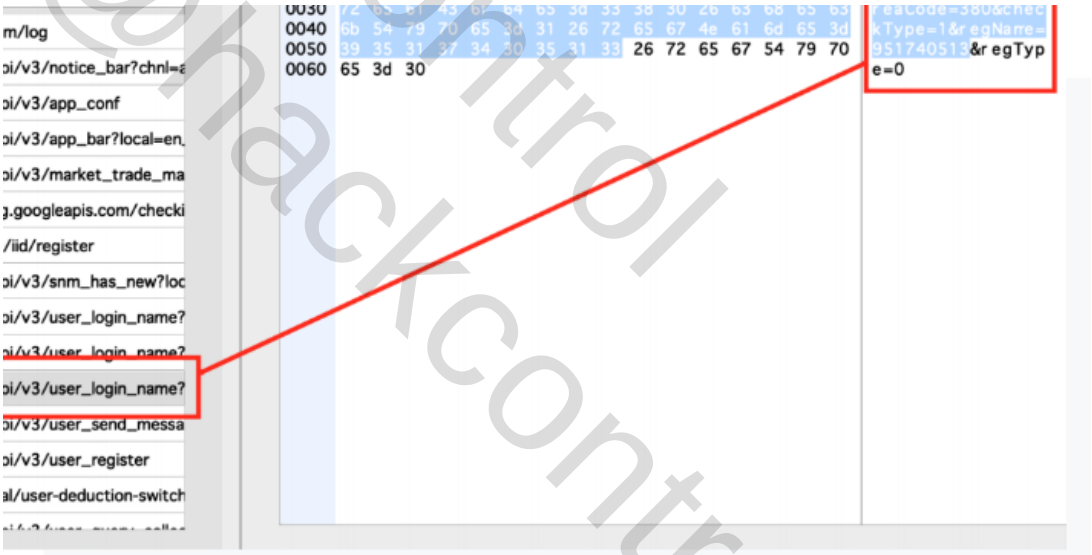
This information can be used by attacker for getting access to users account.

Evidences

Steps to reproduce:

- Sign up/Log in to the application
- Connect to the device with ssh
- Navigate to application's sandbox
- Open Cache.db with any SQLite viewer, from /Library/Caches/com.company.exchange/

For example, areaCode and regName parameters with data, in sum this is user's mobile phone number:



Recommendations	Application shouldn't store requests and responses locally or this database should be encrypted. For example, you can use SQLite 3 library for that. This is C++ wrapper that provides an API for the SQLite commands.

Insecure sending of the user's mobile phone (areacode+regname)

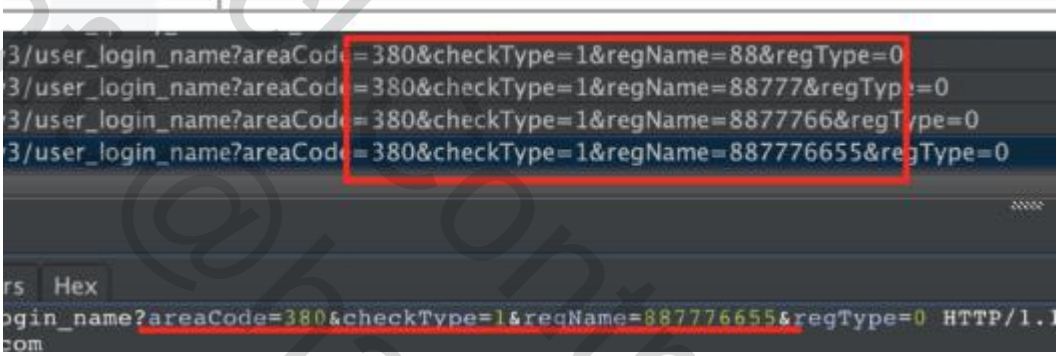
#3	Description	Type: Real

Application sends user's mobile phone number from "Sign up" screen with GET method. RESTful web services should be secured to prevent leaking credentials. Logins, passwords, security tokens, and API keys should not appear in the URL. In POST/PUT requests sensitive data should be transferred in the request body or request headers. In GET requests sensitive data should be transferred in an HTTP Headers.

Evidences

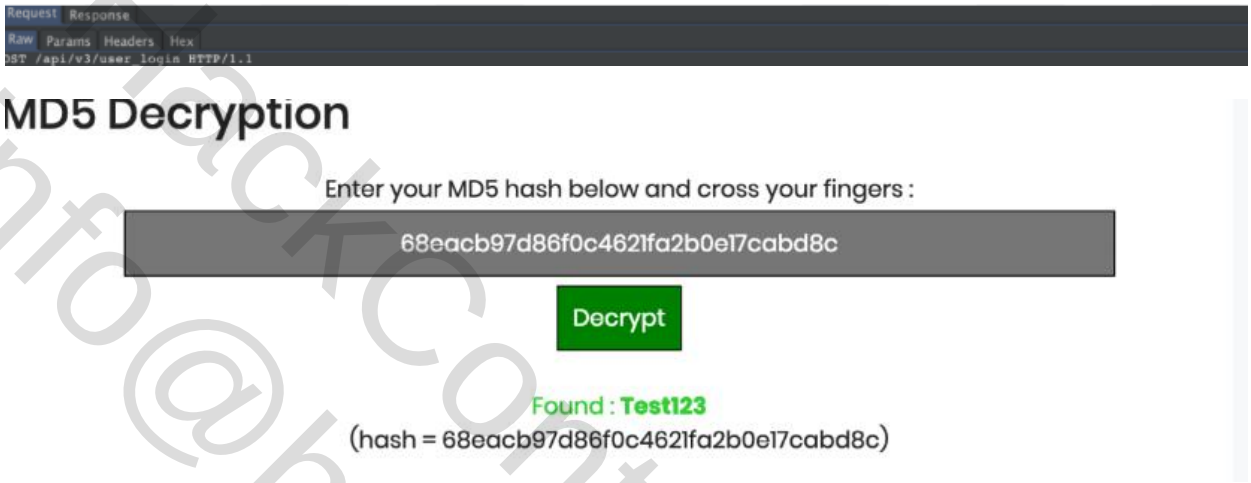
Steps to reproduce:

- Run BurpSuite
- Set up proxy connection on the device
- Install root SSL CA on the device
- Disable certificate validation with SSL KILL SWITCH 2
- Intercept requests from the "Sign up" screen



Recommendations	Remove this requests at all or if it's important for logics - switch them on the POST method for sending sensitive information.

Weak cryptography

#4	Description	Type: Real
In order to exploit this weakness, an adversary must successfully return encrypted code or sensitive data to its original unencrypted form due to weak encryption algorithms or flaws within the encryption process.		
Evidences		
		
Recommendations	Use modern hashing algorithms for example SHA515	

Input fields with sensitive data should be cleared after hiding/opening the application

#5	Description	Type: Real
This is supposed for the password and invite code fields and it will be useful in case when a user sets data in this fields and hides the application without verify/login step.		
Evidences		
Steps to reproduce: - Open the application on the “Sign up”, “Log in” or “Change password” screens - Set password - Hide/Open the application		
Recommendations	The app removes sensitive data from the input fields when backgrounded.	

Clipboard should be disabled for fields with sensitive data

#5	Description	Type: Real
Clipboard is one for all system and sensitive data of our application can be stolen by another one.		
Evidences		
Steps to reproduce: - Open the application on the “Sign up”, “Log in” or “Change password” screens - Select all the text in the password field - Try to copy the text		
Recommendations	Clipboard should be disabled for all the input fields working with sensitive data.	

Application doesn't have jailbreak detection mechanism

#7	Description	Type: Real

Should be implemented functionally independent methods of jailbreak detection and responds to the presence of a jailbroken device by terminating the application or should display Warning pop-up ("Your device appears to be jailbroken. The security of your app can be compromised.") every start.

```
IPhone-6s-Silver:~ root# ipainstaller -l
apreciate.Preciate14
ch.protonmail.vpn
co.vero.app
com.apple.itunesconnect.mobile
com.apple.TestFlight
```

Second jailbreak detection mechanism is Checking file permissions. This mechanism should try to write into location outside of the application's sandbox. This mechanism should try to write into location outside of the application's sandbox. For example, this can be done by having the application attempt to create a file in /private directory.

```
NSError *error;
NSString *stringToBeWritten = @"This is a test.";
[stringToBeWritten
writeToFile:@"/private/jailbreak.txt" atomically:YES
encoding:NSUTF8StringEncoding error:&error];
if(error==nil){
//Device is jailbroken
return YES;
} else {
//Device is not jailbroken
[[NSFileManager defaultManager]
removeItemAtPath:@"/private/jailbreak.txt" error:nil];
}
```

Third jailbreak detection mechanism is Checking protocol handlers. For example, application can attempt to open a Cydia URL. The Cydia app store, which is installed by default by practically every jailbreaking tool, installs the cydia:// protocol handler.

```
if([[UIApplication sharedApplication] canOpenURL:[NSURL
URLWithString:@"cydia://package/com.example.package"]])
{
```

Fourth jailbreak detection mechanism is Calling system APIs. This mechanism should try to calling the system() function with a NULL argument on a non jailbroken device will return "0"; doing the same on a jailbroken device will return "1". This is since the function will check whether /bin/sh can be accessed, and this is only the case on jailbroken devices.

Recommendations	First jailbreak detection mechanism is File-based checks.

Appendix A. Automated Tools

Scope	Tools used
Application Security	Burp Suite ettercap SSL Kill Switch 2 Filza keychain-dumper ipainstaller Needle Log Console Atom DB Browser for SQLite TestSSL Nmap Tested on iPad iOS 11.2.1 with Electra jailbreak