https://e-discovery.ng/

iOS APPLICATION PENETRATION TESTING

Report for:	
Date:	

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

Table of Contents

Executive Summary

E-Discovery (Provider) was contracted by _____ (Client) to carry out an **iOS application** penetration test.

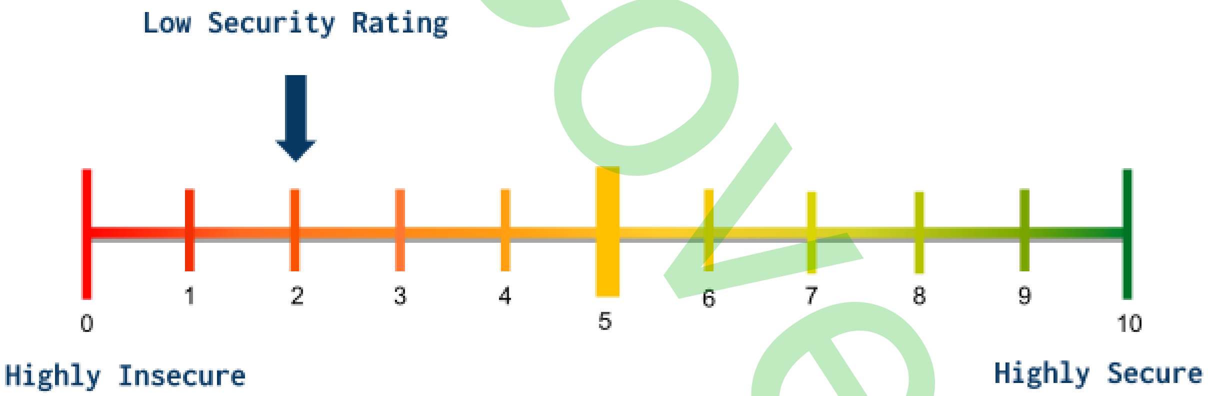
The application provides customers the ability to submit order requests, review design, leave feedback, etc.

The penetration test was conducted between **08.02.2021 - 26.02.2021**.

The penetration test has the following objectives:

- identify technical and functional vulnerabilities
- evaluate a severity level (ease of use, impact on information systems, etc.);
- make a prioritized list of recommendations to address identified weaknesses.

According to our research after performing the penetration testing, the security rating of the client's **iOS application** was identified as **Low**.



Scope

The following list of the information systems was the scope of the penetration testing.

#	Name	Description	Version
1.	__	iOS	

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

Severity	Description
High 🟥🟥🟥🟥	High-level vulnerabilities are easy to exploit and may provide an attacker with complete control of the affected systems, leading 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. Exploits or PoCs aren't available in public access. Exploitation provides only very limited access.
Low 🟩🟩	Low-level vulnerabilities exploitation is extremely difficult, or impact is minimal.
Info 🟦	Information-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.

Summary of Findings

The table below shows the vulnerabilities and their severity. A total of **7 vulnerabilities** were found.

Title	Severity
User’s credential stores locally and not encrypted in application’s sandbox	High
Requests and responses stores insecure in cache.db	High
Insecure sending of the user’s mobile phone (area code+regname)	Medium
Weak cryptography	Medium
Input fields with sensitive data should be cleared after hiding/opening the application	Low
Clipboard should be disabled for fields with sensitive data	Low
Application doesn’t have jailbreak detection mechanism	Info

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 security rating of the client's **iOS application** was identified as **Low**.

The client should pay special attention to the following vulnerabilities:

1. User’s credential is stored locally and not encrypted in the application's sandbox.

Key Findings

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

#1 Description

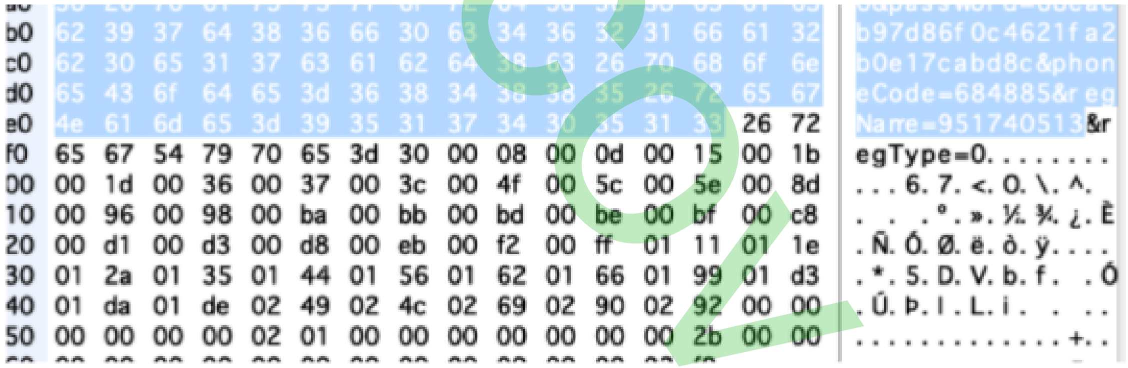
Local database from /var/mobile/Containers/Data/Application/DC6488D9-C54A-4FE8-87DB-49764E92938C/Library/Caches/com.CLIENT.ff stores user’s credentials.

Evidence

Steps to reproduce:

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

Request:



```
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 34 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  . . .°.».½.¾.¿.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;

■■■ Insecure sending of the user's mobile phone (area code+regname)

#2 Description

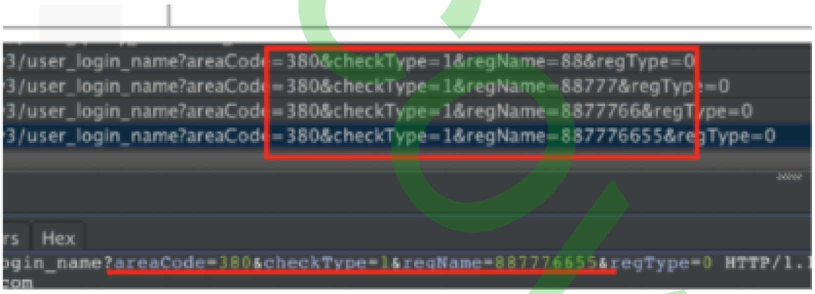
Application sends the user's mobile phone number from the “Sign up” screen with GET method. RESTful web services should be secured to prevent leaking credentials. Logins, passwords, security tokens, and API keys should not appear in the URL. In POST/PUT requests sensitive data should be transferred in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

Evidence

Steps to reproduce:

1. Run Burp Suite
2. Set up proxy connection on the device
3. Install root SSL CA on the device
4. Disable certificate validation with SSL KILL SWITCH 2
5. Intercept requests from the “Sign up” screen

Request:



```
3/user_login_name?areaCode=380&checkType=1&regName=88&regType=0
3/user_login_name?areaCode=380&checkType=1&regName=88777&regType=0
3/user_login_name?areaCode=380&checkType=1&regName=8877766&regType=0
3/user_login_name?areaCode=380&checkType=1&regName=887776655&regType=0
```

```
ogin_name?areaCode=380&checkType=1&regName=887776655&regType=0 HTTP/1.1
com
```

Recommendations

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

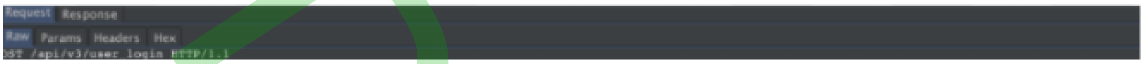
Weak cryptography

#3 Description

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.

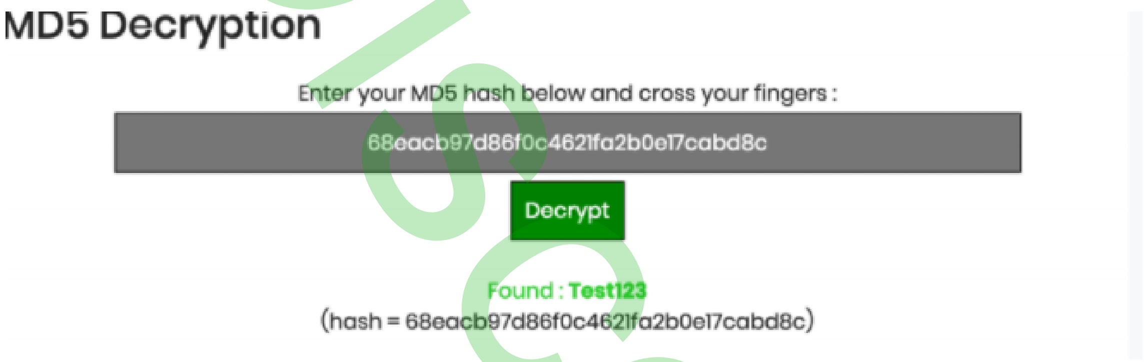
Evidence

Steps to reproduce:



```
OST /api/v3/user_login HTTP/1.1
```

Request:



MD5 Decryption

Enter your MD5 hash below and cross your fingers :

68eacb97d86f0c4621fa2b0e17cabd8c

Found : Test123

(hash = 68eacb97d86f0c4621fa2b0e17cabd8c)

Recommendations

- Use modern hashing algorithms for example SHA515

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

#4 Description

This is supposed for the password and invite code fields and it will be useful in case when a user sets data in these fields and hides the application without a verification/login step.

Evidence

Steps to reproduce:

1. Open the application on the “Sign up”, “Log in” or “Change password” screens
2. Set password
3. 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

Clipboard is one for all systems and sensitive data of our application can be stolen by another one.

Evidence

Steps to reproduce:

1. Open the application on the “Sign up”, “Log in” or “Change password” screens
2. Select all the text in the password field
3. Try to copy the text

Recommendations

- clipboard should be disabled for all the input fields working with sensitive data.

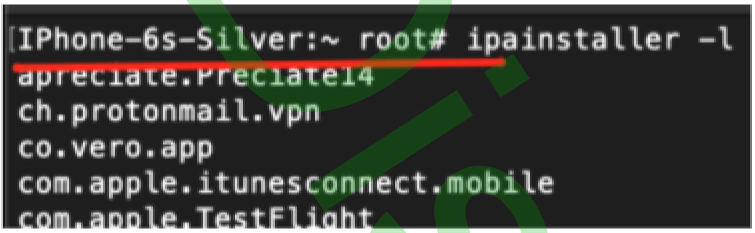
■ Application doesn't have jailbreak detection mechanism

#6 Description

Should be implemented functionally independent methods of jailbreak detection and respond 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.

Evidence

Request:

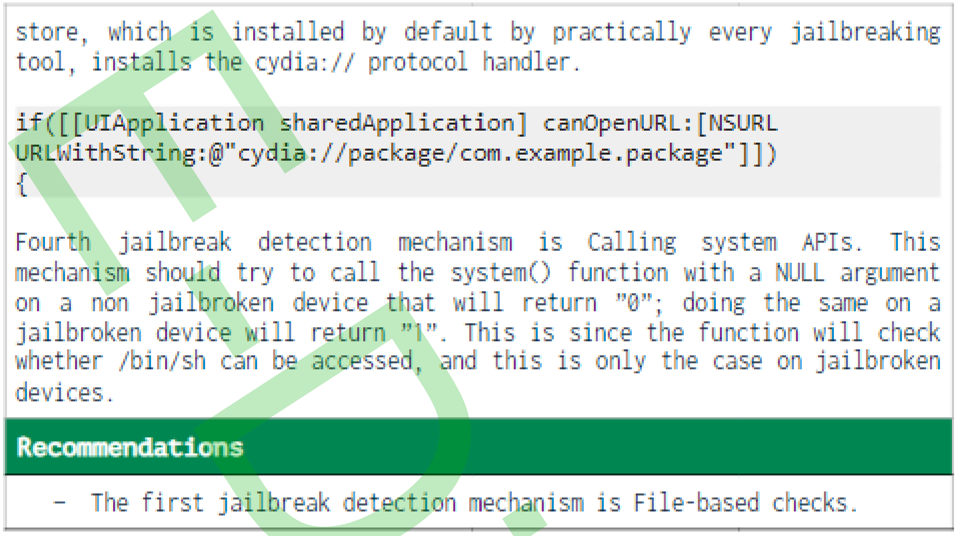


```
[IPhone-6s-Silver:~ root# ipainstaller -l
apreciate.Preciate14
ch.protonmail.vpn
co.vero.app
com.apple.itunesconnect.mobile
com.apple.TestFlight
```

Second jailbreak detection mechanism is Checking file permissions. This mechanism should try to write into locations outside of the application's sandbox. This mechanism should try to write into locations 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, an 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 call the system() function with a NULL argument on a non jailbroken device that 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

- The 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