Kok Sang Teo Follow

Jun 1, 2021 · 5 min read · Listen

Save

Yet another password manager app, how to better secure it?

Vulnerability Disclosure timeline

10 Dec 2020 — Reported to developer via Github channel for responsible disclosure

18 Jan 2021 — Pinged developer again for the disclosure; no response

2 Jun 2021 — Technical advisory published in Medium

2 Jun 2021 — Informed MITRE; Awaiting response from MITRE

11 Nov 2021 — MITRE responded. CVE reserved; CVE-2021–3179

Update — Technical Advisory: Insecure data storage resulting to authentication bypass (CVE-2021–3179)

Vulnerability Type: Insecure Data Storage (MSTG-STORAGE-1 and MSTG-STORAGE-2)

Summary: GGLocker iOS application, contains an insecure data storage of the password hash value which resulted in authentication bypass.

Impact: Authentication bypass of the password vault by extracting the hash value of the stored password, decrypting the hash to recover plaintext password and/or manipulating the return value to gain access to the digital locker vault.

Mobile penetration testing is a career where one evaluate the security of a particular mobile application in smartphone platforms such as iOS, Android etc.

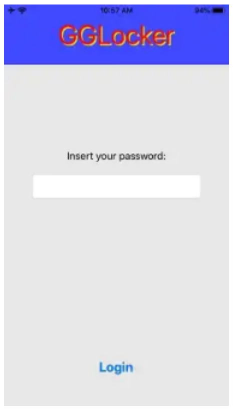
While I was reading up on OWASP MSTG (you guys should really take a good read on their resources), I have learn the importance of having a mobile application that is secure and yet functional for users. Hackers can always find a way or two to extract sensitive information from a mobile device and/or mobile application and use it for nefarious deeds.

Back in Dec 2020, I stumbled upon a particular open sourced password manager application (app) that is available in the iOS app store for free to download. The issue is, what if the password manager app is vulnerable?

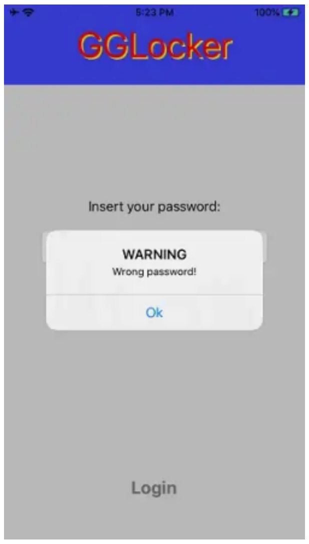
Insecure Data Storage found in GGLocker password manager app resulting in authentication bypass of the app

Password managers apps are an attractive target for hackers as they contain a users' login credentials. This means that if you can break it, you gain access to a user's credentials stored within. It is therefore important for the mobile app to be security conscious when handling the master password to access the vault of crown jewels.

When you first download the app, it will prompt you to enter a password which then create your own credentials database (the vault). After which, you will then proceed to enter the password you have entered to access the vault as shown below.



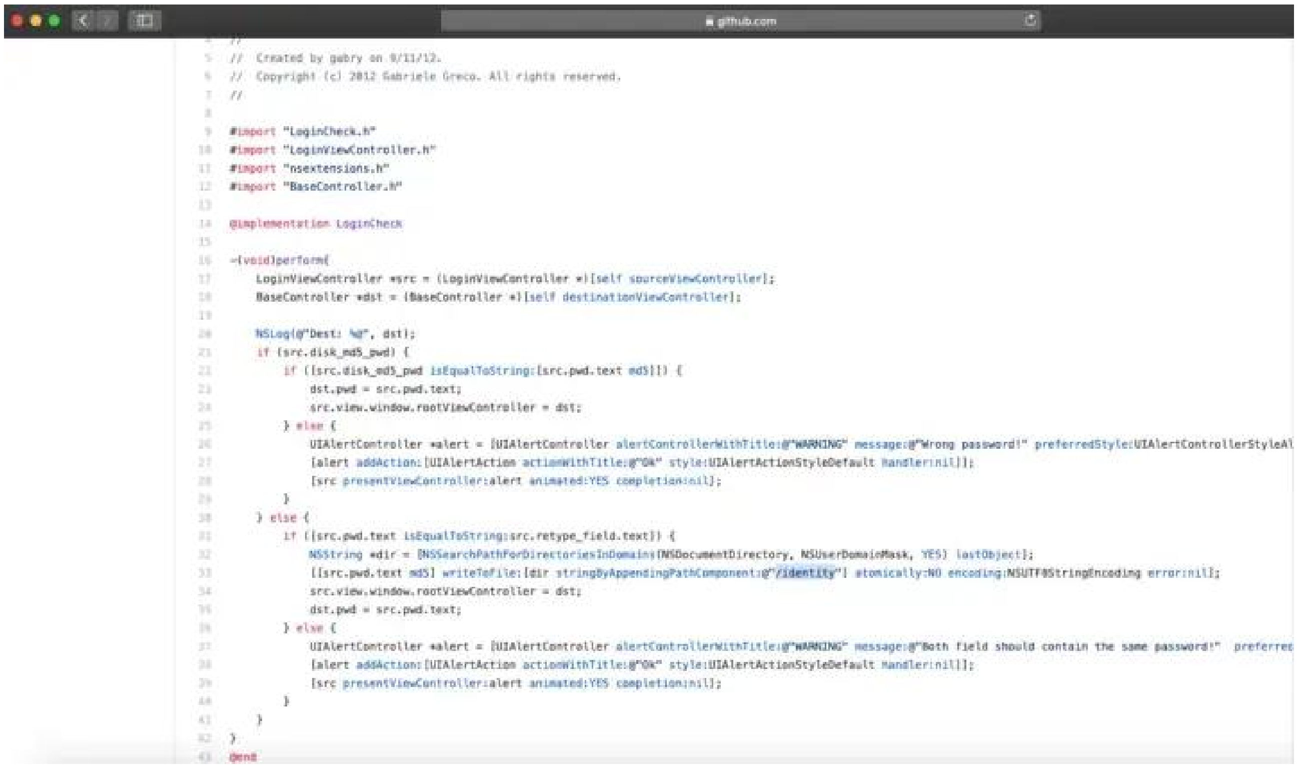
Entering the wrong password will deny access to the password vault and entering the correct password will allow access to the password vault.



The main issue that I found is on the insecure storage of the password that is being used to access the vault.

Details: By inspecting the app's source code found in the developer's Github page, we know that the app does not store the password in plaintext. The text value is being hashed and the app stores the hash value.

We also identified that the LoginViewController has a function called perform that is responsible to validate the access to the vault. The perform function checks if the hash value the enter is identical to the hash value of the initial password set as shown below:

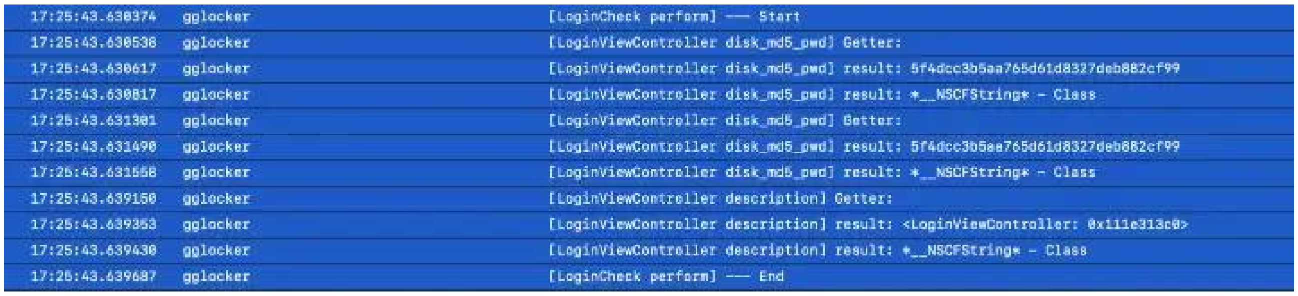


```
//
//  Created by gabry on 9/11/12.
//  Copyright (c) 2012 Gabriele Greco. All rights reserved.
//

#import "LoginCheck.h"
#import "LoginViewController.h"
#import "nsextensions.h"
#import "BaseController.h"

@implementation LoginCheck

-(void)perform{
    LoginViewController *src = (LoginViewController *)[self sourceViewController];
    BaseController *dst = (BaseController *)[self destinationViewController];

    NSLog(@"Dest: %@", dst);
    if (src.disk_md5_pwd) {
        if ([src.disk_md5_pwd isEqualToString:[src.pwd.text md5]]) {
            dst.pwd = src.pwd.text;
            src.view.window.rootViewController = dst;
        } else {
            UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"WARNING" message:@"Wrong password!" preferredStyle:UIAlertControllerStyleAl
            [alert addAction:[UIAlertAction actionWithTitle:@"Ok" style:UIAlertActionStyleDefault handler:nil]];
            [src presentViewController:alert animated:YES completion:nil];
        }
    } else {
        if ([src.pwd.text isEqualToString:src.retype_field.text]) {
            NSString *dir = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) lastObject];
            [[src.pwd.text md5] writeToFile:[dir stringByAppendingPathComponent:@"/identity"] atomically:NO encoding:NSUTF8StringEncoding error:nil];
            src.view.window.rootViewController = dst;
            dst.pwd = src.pwd.text;
        } else {
            UIAlertController *alert = [UIAlertController alertControllerWithTitle:@"WARNING" message:@"Both field should contain the same password!"  preferred
            [alert addAction:[UIAlertAction actionWithTitle:@"Ok" style:UIAlertActionStyleDefault handler:nil]];
            [src presentViewController:alert animated:YES completion:nil];
        }
    }
}
@end
```

source code of "perform" function

As a hacker, this means that if I tamper the hash value of the perform function to an attacker controlled value, it is possible to perform an authentication bypass to access the vault! **I believe that using a non jailbroken iPhone with Frida, Theos or Grapefruit would be able to perform the exploit.** For demonstration purpose, I will be using Theos to perform hooking of LoginViewController to inspect the value of the return hashed value.

First, we will perform a classdump extraction of the mobile app to identify the function and we will then hook on the function to inspect the return value. The result should look something like this when inspecting app logs from macOS Console:



```
17:25:43.630374   gglocker                [LoginCheck perform] --- Start
17:25:43.630536   gglocker                [LoginViewController disk_md5_pwd] Getter:
17:25:43.630617   gglocker                [LoginViewController disk_md5_pwd] result: 5f4dcc3b5aa765d61d8327deb882cf99
17:25:43.630817   gglocker                [LoginViewController disk_md5_pwd] result: *__NSCFString* - Class
17:25:43.631301   gglocker                [LoginViewController disk_md5_pwd] Getter:
17:25:43.631490   gglocker                [LoginViewController disk_md5_pwd] result: 5f4dcc3b5aa765d61d8327deb882cf99
17:25:43.631568   gglocker                [LoginViewController disk_md5_pwd] result: *__NSCFString* - Class
17:25:43.639150   gglocker                [LoginViewController description] Getter:
17:25:43.639353   gglocker                [LoginViewController description] result: <LoginViewController: 0x111e313c0>
17:25:43.639430   gglocker                [LoginViewController description] result: *__NSCFString* - Class
17:25:43.639687   gglocker                [LoginCheck perform] --- End
```

We noticed that "disk_md5_pwd" equals to "5f4dcc3b5aa765d61d8327deb882cf99" which translates to "password". This also means that if a hacker enter the word "password", the individual would be able to access the vault.

A skilled hacker can also take this vulnerability to the next level by manipulating the return value to a particular value of the individual choice :/

Recommendations:

1. We can consider to implement jailbreak detection function to prevent the app from running in a jailbroken phone. By doing so, it makes things harder for a hacker to perform exploitation on the app. ***However do note that the vulnerability can still be performed in a non jailbroken environment (just that it will be much harder).***
2. To consider to store the hash value in a keychain to prevent unauthorized tampering of the value.

> *The purpose of the article is to educate and spread awareness on the importance of secure coding to better secure a mobile app.* ***In no way*** *I am promoting unethical hacking. Please hack responsibly and feel encouraged that you are taking baby steps in making the digital world more secure :)*