Baby Boom of Coding Kiddies Considered Harmful

Saisi Peter

November 18, 2014

Abstract

This paper investigates The Ropes - a popular Dining Application for Dartmouth students made by Dartmouth students. The latter group considered harmful. Using Ropes, we will examine how to not write iOS applications by demonstrating how to pilfer and extract instances of sensitive information from the app. After which, I give suggestions on how to secure apps and include working code of a sane and tamed application-server interaction model. This investigation is important because the App Store has lowered barriers of entry and distribution leading to a proliferation of equally wanting and insecure applications. As a consequence, many new and potential developers need to be aware of the additional effort they should incur in creating secure consumer facing applications.

1 Introduction

The Ropes allows students to look up their meal plan balance. A student gives the app their login credentials for `www.dartmouth.managemyid.com`. Unfortunately, the app does nothing to protect students' personal login credentials. In its temerity to be as insecure as `Windows 98`, it also exposes private developer information. I was able to steal my own password, extract my dining history, mine persistent cookies and discover the developer's oauth keys and secrets for their Facebook and Twitter accounts. For each vulnerability, I provide ways of mitigating or eliminating the threat.

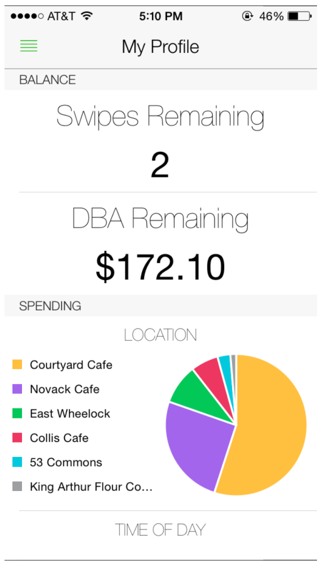


Figure 1: Screenshot of Ropes

2 Prior Work

2.1 What others have done

Other people have researched and presented papers on ways of compromising iOS applications. Adam Kliarsky wrote a great paper on how to pentest iOS apps [1]. Blackhat usually has regular sessions on compromising iOS apps. [2]

2.2 Relationship with prior work

My work is *complementary* because it builds on the existing corpus of knowledge on how to poke apps for holes, but it is also *different* because this is a very specific examination of a Dartmouth student-targeted application. This is the first time Ropes is being audited and I examine the specific security vulnerabilities and implications that arise by virtue of it being a Dartmouth-student targeted application.

[1]http://www.sans.org/reading-room/whitepapers/testing/ipwn-apps-pentesting-ios-applications-34577
[2]http://www.blackhat.com/us-13/training/ios-application-hacking%E2%80%93pentesting-mobile-apps.html

3 What I Did

3.1 Overview

I devised a threat model that encompassed the various forms of attack Ropes could face. For each threat model, I examine as fit: the vulnerability, information at risk, how to obtain the information, how the information might be abused, and how to protect against or mitigate the vulnerability.

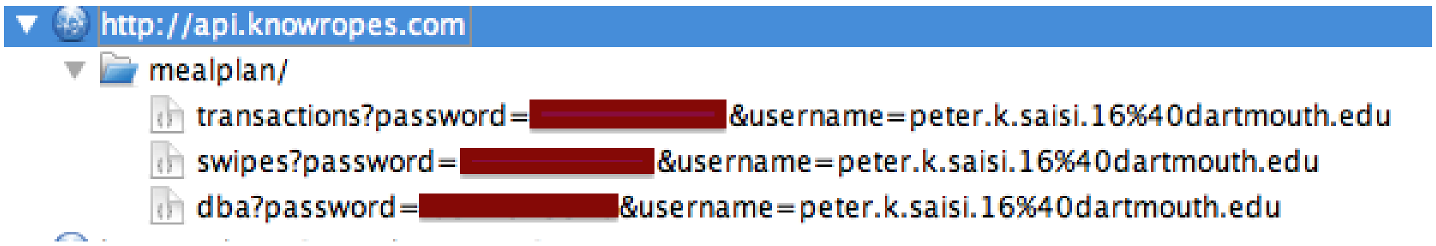
3.1.1 Threat Model

I consider two kinds of attackers:

- A remote attacker. The attacker can access information in transit to and from app.
- A local attacker. The attacker can attack the:
  - iDevice i.e iPhone, iPad
  - Trusted computing device that holds the iDevice backups i.e: computer configured to iTunes sync with iDevice

3.1.2 Remote Attacker

Ropes communicates with `api.knowropes.com` for every session. To examine what kind of information is transmitted, I used `Charles App` [3] to sniff traffic. Using my MacBook as a proxy, I was able to intercept traffic being relayed to the server end point. Here's data captured from one refresh of the app:



```
http://api.knowropes.com
  mealplan/
    transactions?password=[redacted]&username=peter.k.saisi.16%40dartmouth.edu
    swipes?password=[redacted]&username=peter.k.saisi.16%40dartmouth.edu
    dba?password=[redacted]&username=peter.k.saisi.16%40dartmouth.edu
```

Figure 2: Sniffed data, unfortunate

3.1.3 Mistakes

- Information such as the password is submitted in **plaintext** as a **GET** request.
- There is no use of encryption such as SSL.
- Multiple concurrent requests with user credentials.

3.1.4 Implications

- The lack of even basic encryption or SSL means that credentials can be easily obtained using sniffing tools i.e Wireshark. An even more determined attacker can easily employ APN spoofing to trick the app into thinking its connected to a legitimate carrier and relay plaintext information to the attacker.

[3]www.charlesproxy.com

- The use of GET requests exposes the user's credentials to great insecurity. A GET request means that the full url being requested i.e `api.knowropes.com/mealplan/transactions?password=[redacted]...` is logged on the server. The string credentials in the url can also be exposed by HTTP Proxies. Even with SSL enabled, tools like `Bluecoat` can retrieve GET strings. The GET method is meant for data retrieval and should not have any side-effects. Somewhere on the knowropes.com is a nginx server is a log with everyone's passwords waiting to be compromised.

- Having three separate connections (loaded with credentials) to retrieve information that could be retrieved in one connection speaks of bad API design. It also gives the attacker more opportunity for interception since every single time the app is refreshed there are thrice as many requests.

- 50% of the students I surveyed mentioned that their managemyid password is similar to their banner password. This is an incredible security risk because with that password, an attacker can do anything from add/drop courses, change D-PLAN, view transcripts, submit course evaluations or access student email.

3.1.5 Fixing Mistakes

- A better well thought out API design. This is the current Ropes model:

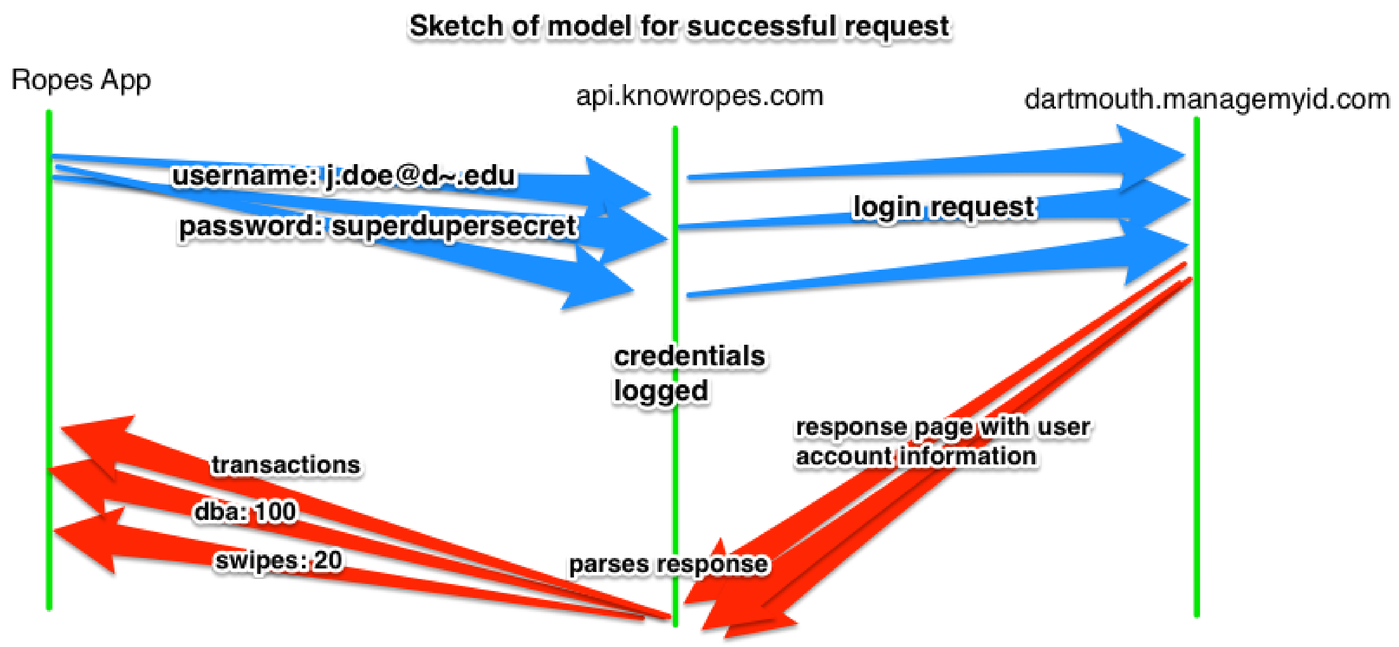


Figure 3: Why would you do this?

From the current model where the username and password relayed to the server for every single request, we can infer that the developers' original intent was to not have student credentials persist on server. However, the use of GET requests to relay information mean that the query strings are logged or easily exposed to sniffing tools even when using SSL. The current model is a Rube Goldeberg contraption that creates way too many opportunities for interception or man-in-the-middle attacks.

A better model would be:

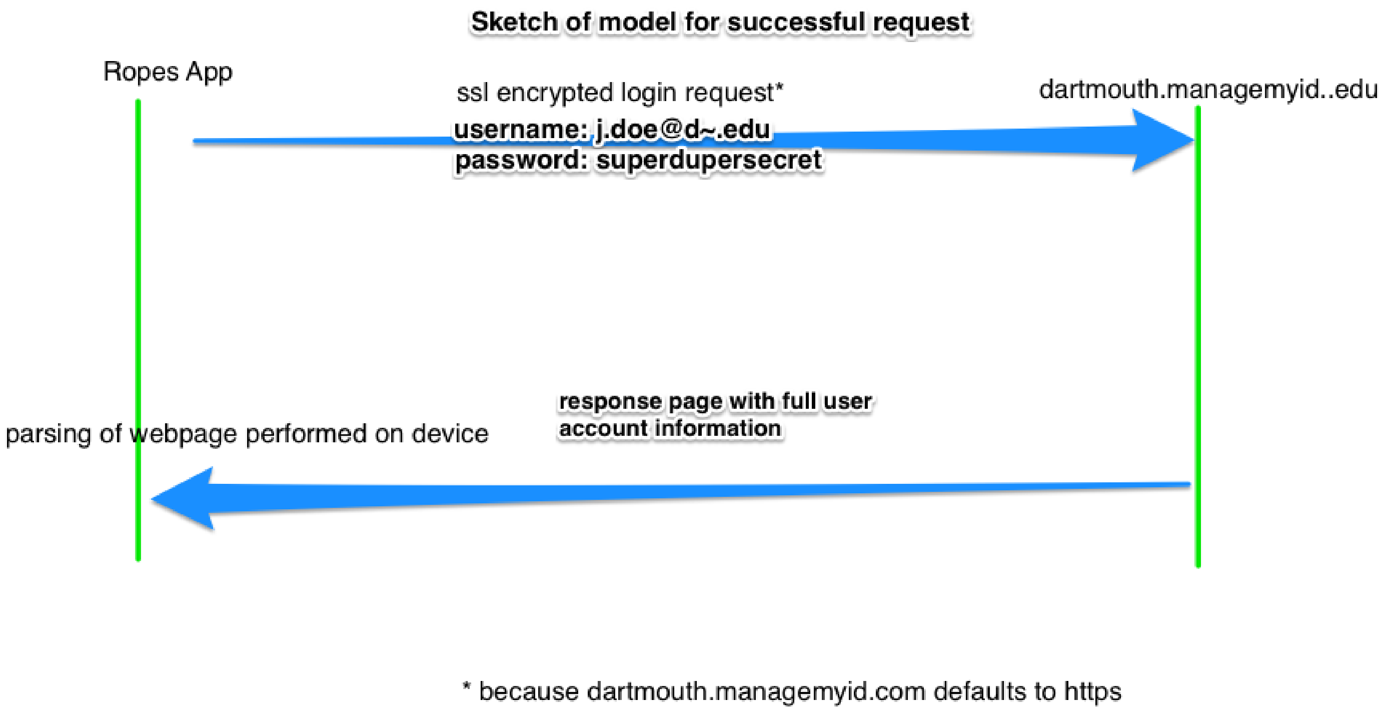
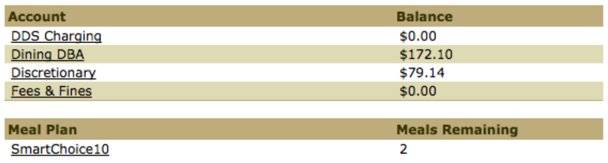

Sketch of model for successful request
Ropes App
ssl encrypted login request*
username: j.doe@d~.edu
password: superdupersecret
dartmouth.managemyid..edu
response page with full user account information
parsing of webpage performed on device
* because dartmouth.managemyid.com defaults to https

Figure 4: This makes me happy

This model entirely eliminates the ropes server. There is no viable reason for server side parsing when the managemyid webpage is only `10 Kilobytes`. The parsing can be delegated to the mobile device. There is use of only one login connection request since a successful login gives you access to all required information. Here's a screenshot of an authenticated login that can be parsed.



Account	Balance
DDS Charging	$0.00
Dining DBA	$172.10
Discretionary	$79.14
Fees & Fines	$0.00

Meal Plan	Meals Remaining
SmartChoice10	2

Figure 5: managemyid dashboard

All the information that could be desired is available in one landing page. I made a sample Android application to demonstrate this model[4]. There is very little device overhead experienced by shifting the parsing to it. I have included some portions of code in the appendix.

- If the developers still insist on maintaining the man-in-the-middle server, all requests sent from the app should be POST requests over a secured channel i.e: encrypted data or using SSL.

[4]https://github.com/Saisi/DartmouthAndroidDining

3.1.6 Local Attacker

I installed the latest version of Ropes `(v1.0.1)` on a jailbroken[5] `iPad 2` running `iOS 7.1.1` . The following Cydia packages were used:

- `Class Dump` - extracts Objective-C `.h` interface definitions from binary.
- `Cycript` - a runtime execution server and disassembler. It allows a person to attach to a running process and operate within that process' runtime. Methods can be invoked or overridden.
- `PanGu` - jailbreak utility.
- `dumpdecrypted` - generates a code injecting library for dumping of decrypted binary.

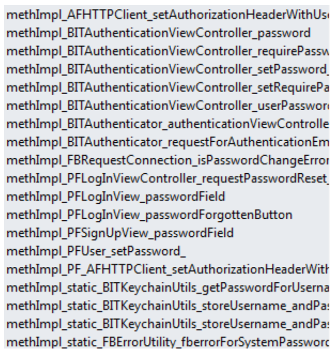
Details and resources on how to initiate the following are found in the appendix: Jailbreak, Install Cydia packages, and Decrypt binary files.

3.1.7 Static Binary Analysis

App Store binaries are signed by both developer and Apple such that the binary is encrypted. However, when an app is opened, it is decrypted before being loaded into memory. I used `dumpdecrypted` to exploit this fact by letting the kernel load the binary and then dump the decrypted in-memory binary to a file for analysis.

With the static dump of decrypted binary, I used `Hopper Disassembler`[6] to reverse engineer the code. The disassembled code provides insights into:

- Application logic: Hopper goes the further mile and constructs pseudocode from assembly
- Method names:



```
methImpl_AFHTTPClient_setAuthorizationHeaderWithUs
methImpl_BITAuthenticationViewController_password
methImpl_BITAuthenticationViewController_requirePassw
methImpl_BITAuthenticationViewController_setPassword_
methImpl_BITAuthenticationViewController_setRequirePa
methImpl_BITAuthenticationViewController_userPasswor
methImpl_BITAuthenticator_authenticationViewControlle
methImpl_BITAuthenticator_requestForAuthenticationEm
methImpl_FBRequestConnection_isPasswordChangeError
methImpl_PFLogInViewController_requestPasswordReset_
methImpl_PFLogInView_passwordField
methImpl_PFLogInView_passwordForgottenButton
methImpl_PFSignUpView_passwordField
methImpl_PFUser_setPassword_
methImpl_PF_AFHTTPClient_setAuthorizationHeaderWith
methImpl_static_BITKeychainUtils_getPasswordForUserna
methImpl_static_BITKeychainUtils_storeUsername_andPa
methImpl_static_BITKeychainUtils_storeUsername_andPa
methImpl_static_FBErrorUtility_fberrorForSystemPasswor
```

Figure 6: method names

- Hard coded secret keys:

```
cfstring_[redacted]jH
```

```
cfstring_3[redacted]2f
```

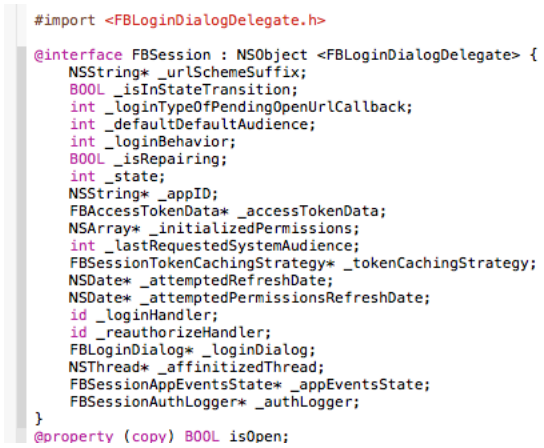
[5]A jailbreak is a form of privilege escalation on iOS devices that breaks Apple imposed limitations
[6]http://www.hopperapp.com/download.html

3.1.8 Runtime Manipulation

Using `Cycript`, I was able to obtain header files for Ropes. Header files are great giveaways on the structure and organisation of code because they include instance variables and method declarations. The full list of header files can be obtained at `https://github.com/Saisi/ConsideredHarmful/tree/master/headers`. Potentially interesting header files obtained include:

- `FBSession.h` - outlines the structure of Facebook data sessions. We could gain information such as oauth tokens and secrets
- `LoginVerifyStudentViewController.h` - a template of login logic
- `NSCopying.h` - NSCopying can be swizzled and have copy actions overridden with a custom copy that dumps objects to a file. It can be made to dump copied strings

For demonstration, I use `FB.Session.h`. Here's a snippet

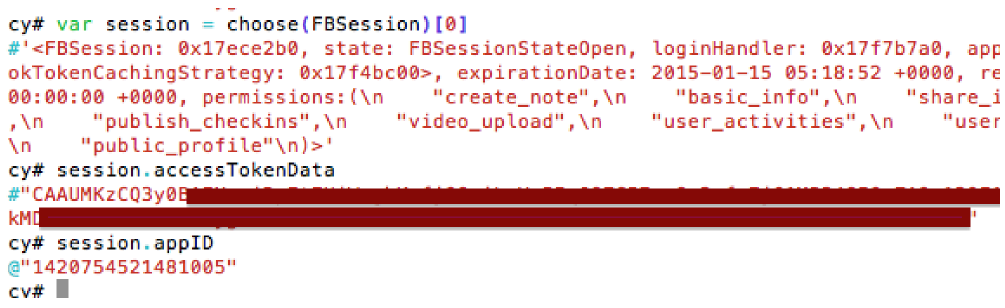


```
#import <FBLoginDialogDelegate.h>

@interface FBSession : NSObject <FBLoginDialogDelegate> {
    NSString* _urlSchemeSuffix;
    BOOL _isInStateTransition;
    int _loginTypeOfPendingOpenUrlCallback;
    int _defaultDefaultAudience;
    int _loginBehavior;
    BOOL _isRepairing;
    int _state;
    NSString* _appID;
    FBAccessTokenData* _accessTokenData;
    NSArray* _initializedPermissions;
    int _lastRequestedSystemAudience;
    FBSessionTokenCachingStrategy* _tokenCachingStrategy;
    NSDate* _attemptedRefreshDate;
    NSDate* _attemptedPermissionsRefreshDate;
    id _loginHandler;
    id _reauthorizeHandler;
    FBLoginDialog* _loginDialog;
    NSThread* _affinitizedThread;
    FBSessionAppEventsState* _appEventsState;
    FBSessionAuthLogger* _authLogger;
}
@property (copy) BOOL isOpen;
```

Figure 7: FB.Session.h

The most interesting information is held in the `accessTokenData` instance variable. After hooking Cycript to the running instance of Ropes, I proceed as pictured below



```
cy# var session = choose(FBSession)[0]
#'<FBSession: 0x17ece2b0, state: FBSessionStateOpen, loginHandler: 0x17f7b7a0, app
okTokenCachingStrategy: 0x17f4bc00>, expirationDate: 2015-01-15 05:18:52 +0000, re
00:00:00 +0000, permissions:(\n    "create_note",\n    "basic_info",\n    "share_i
,\n    "publish_checkins",\n    "video_upload",\n    "user_activities",\n    "user
\n    "public_profile"\n)>'
cy# session.accessTokenData
#"CAAUMKzCQ3y0B[redacted]
kM[redacted]'
cy# session.appID
@"1420754521481005"
cy# 
```

I used Cycript's choose method to locate an `FBSession` object instance in the app's runtime memory. After which, I accessed the `accessTokenData`. Very easy. Very effective.

3.1.9 Mistakes

- Tokens in human-readable form

- Limited use of encryption
- Memory unsecured
- Methods and class names not obfuscated

3.1.10 Implications

- Hard coded tokens are low hanging fruit for an attacker. Easy to identify and retrieve.
- Critical data stored in instance variables are easily referenced. It is able to obtain the developers tokens for Parse, Facebook, Twitter and Hockey app. Using the above demonstrated Facebook app tokens, I was able to authenticate an application as The Ropes Facebook app. A malicious attacker might use this opportunity to sign malicious Facebook apps using Ropes tokens leading to application suspension.
- The lack of obfuscation makes it obvious to an attacker where to target. `setPassword(arg)` is a dead ringer unlike `kKHKaflkdsKJHFD(arg)`.

3.1.11 Fixing Mistakes

These suggestions were influenced by J. Zdziarski's work: [7]

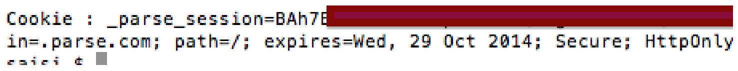
- Hard code session keys in non humanly readable format. The encrypted session keys can be hard coded and only decrypted when needed and then disposed off.
- Secure memory. Critical data persisted in instance variables even after they had outlived its purpose. Overwrite or deallocate memory for sensitive credentials after they are used.
- Implement anti-tamper mechanisms. Even the best mechanisms can be undone, however, there needs to be protection against easy attacks thus raising the bar. Suggestions include:
  - False contacts. False contacts bait the attacker into calling methods that seem appealing. This misdirection makes cracking harder and frustrating enough to stump an impatient cracker.
  - Process trace checking. When an application is being debugged, the kernel sets the P_TRACED flag. This helps detect when gdb or a debugging mechanism is attached to a process.
  - Runtime integrity checks. Before a method is executed, it can be reviewed to determine whether it has been altered.
- Implement tamper response mechanisms. When the application realises it is being accessed in a nefarious way it could:
  - Wipe critical data
  - Disable network access
  - Alert user/developer via email
- Encrypt or obfuscate class names.
- Check often hijacked classes such as NSString that are usually hijacked to dump application memory.
- Stripping to remove unneeded symbols from symbol table making it more difficult for an attacker to piece together what's happening in the code.
- Jailbreak detection. Check for existence of jailbreak files and sandbox integrity.

[7]http://www.zdziarski.com/blog/

3.1.12 Attacking the backup

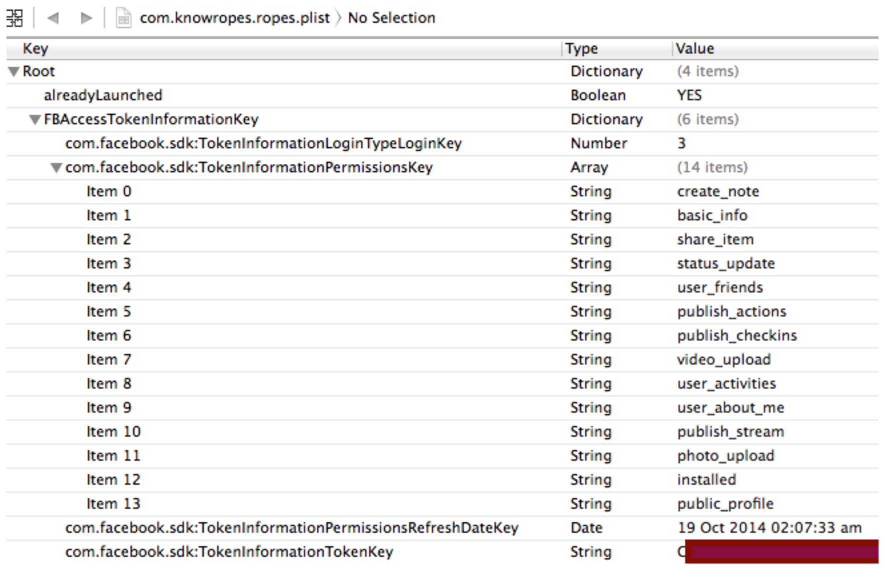
If a user maintains an unencrypted iDevice backup on a computer, the user's personal details can still be gleaned from the iTunes backup. Using `iPhoneBackUpExtractor`[8], I was able to obtain the following files:

- *Cookies.binarycookies* Contains a persistent cookie for an authenticated `parse` session.



```
Cookie : _parse_session=BAh7E[redacted]
in=.parse.com; path=/; expires=Wed, 29 Oct 2014; Secure; HttpOnly
```

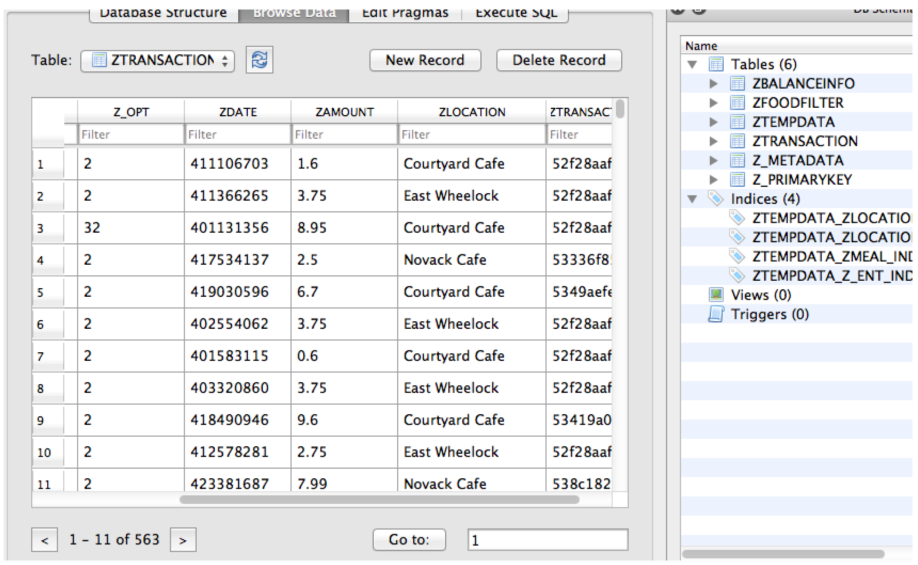
- *com.knowropes.ropes.plist* Contains a Facebook token key.



com.knowropes.ropes.plist 〉 No Selection

Key	Type	Value
▼Root	Dictionary	(4 items)
alreadyLaunched	Boolean	YES
▼FBAccessTokenInformationKey	Dictionary	(6 items)
com.facebook.sdk:TokenInformationLoginTypeLoginKey	Number	3
▼com.facebook.sdk:TokenInformationPermissionsKey	Array	(14 items)
Item 0	String	create_note
Item 1	String	basic_info
Item 2	String	share_item
Item 3	String	status_update
Item 4	String	user_friends
Item 5	String	publish_actions
Item 6	String	publish_checkins
Item 7	String	video_upload
Item 8	String	user_activities
Item 9	String	user_about_me
Item 10	String	publish_stream
Item 11	String	photo_upload
Item 12	String	installed
Item 13	String	public_profile
com.facebook.sdk:TokenInformationPermissionsRefreshDateKey	Date	19 Oct 2014 02:07:33 am
com.facebook.sdk:TokenInformationTokenKey	String	C[redacted]

- *TheRopes.sqlite* Complete database dump of meals, cost and locations where they took place.



	Z_OPT	ZDATE	ZAMOUNT	ZLOCATION	ZTRANSAC
1	2	411106703	1.6	Courtyard Cafe	52f28aaf
2	2	411366265	3.75	East Wheelock	52f28aaf
3	32	401131356	8.95	Courtyard Cafe	52f28aaf
4	2	417534137	2.5	Novack Cafe	53336f8
5	2	419030596	6.7	Courtyard Cafe	5349aefe
6	2	402554062	3.75	East Wheelock	52f28aaf
7	2	401583115	0.6	Courtyard Cafe	52f28aaf
8	2	403320860	3.75	East Wheelock	52f28aaf
9	2	418490946	9.6	Courtyard Cafe	53419a0
10	2	412578281	2.75	East Wheelock	52f28aaf
11	2	423381687	7.99	Novack Cafe	538c182

[8]http://www.iphonebackupextractor.com/

3.1.13 Mistakes

- Aggressive inclusion of credentials into backup.
- No protection/encryption of sqlite database.

3.1.14 Implications

- A user's privacy is compromised since a complete database of meals, price, locations is deposited unprotected.
- Leakage of tokens i.e. Facebook, Parse credentials. An attacker can hijack or impersonate an account or session by piecing such information.

3.1.15 Fixing Mistakes

- Exercise greater discretion in choosing what to backup.
- Always be on the offence. Encrypt the database. Never rely on the user being proactive and prudent.

3.2 Open Challenges - What might be done in future work?

An open challenge is taking the fight to the server. As of November 18th 2014, the `api.knowropes.com` server was running on `nginx/1.4.4` which is a set of revisions behind the most up to date version `nginx/1.7.7`. `nginx/1.4.4` is currently vulnerable [9] to:

- Request line parsing vulnerability (CVE-2013-4547)
- SPDY heap buffer overflow (CVE-2014-0133)
- STARTTLS command injection (CVE-2014-3556)
- SSL session reuse vulnerability (CVE-2014-3616)

The challenge would be to chain some of the vulnerabilities to access logfiles of user credentials or intercept credentials being relayed through the server.

4 Conclusion

The huge recent emphasis on learning how to code combined with the popularity of the App Store has led to a great interest and explosion of mobile apps. The accessibility of online tutorials, documentation and tools has lowered the barriers of entry for developing mobile software. After Dong Nguyen deleted his Flappy Bird app from the App Store, the store was flooded with one Flappy Bird clone every **24** minutes[10]. Before Apple threw down the gauntlet, there were **2000** fart apps in the App Store[11]. These low barriers of entry encourage both the incumbent and novice programmers to try their hands at app making. Unfortunately, this mix of rush and inexperience leads to a compromise on quality.

Dartmouth Students: Change Your Passwords.
Developers: Curb Your Enthusiasm.

An ode to everyone: Baby Boom of Coding Kiddies Considered Harmful.

[9] http://nginx.org/en/security_advisories.html
[10] http://www.forbes.com/sites/insertcoin/2014/03/06/over-sixty-flappy-bird-clones-hit-apples-app-store-every-single-day/
[11] http://techland.time.com/2011/05/24/apple-has-approved-500000-mobile-apps-1123-fart-apps/

5 Credits

I would like to thank Professor Charles Palmer for the instruction and great CS55 class that spurred this research. I would also like to thank fellow classmates for their great discussion and input on Piazza.

6 Disclosure

I notified The Ropes developers in advance before I presented this paper. As of yet, I have not heard from them.

References

[1] Jonathan A. Zdziarski *Hacking and Securing iOS Applications* 2012.

This is not the last page

6.1 Appendix

6.2 Tools

6.2.1 Cydia Apps

Install Cycript:

- Change Cydia mode to expert
- Search ‘cycript’

Install Class Dump:

- Change Cydia mode to expert
- Search ‘Class Dump’

Install and use dumpdecrypted:

- On your computer, open terminal
- git clone https://github.com/stefanesser/dumpdecrypted
- cd dumpdecrypted
- make
- you should now have dumpdecrypted.dylib in the current directory
- transfer dumpdecrypted.dylib to your iDevice
- ssh into your iDevice
- DYLD_INSERT_LIBRARIES=dumpdecrypted.dylib
- locate the directory that has ropes then run
- /var/mobile/Applications/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx/TheRopes.app/TheRopes mach-o decryption TheRopes.decrypted
- TheRopes.decrypted is your decrypted binary

Attack to process using cycript:

- ps aux — grep -i process_name_here
- get the process id for the process corresponding to Ropes
- cycript -p use_process_id_here

6.2.2 Jailbreaking Resources

To get started with jailbreaking and download Pangu, visit `http://en.pangu.io/`
For answers to most jailbreak questions, `http://www.jailbreakqa.com/`

6.2.3 Sample code

This is the parser function from my sample app at `https://github.com/Saisi/DartmouthAndroidDining`. It's only 36 lines long. Considering how expressive `Java` is, this could be reduced to 10-20 in any less verbose programming language or a bash one-liner **:)**

```java
public static HashMap<String, Double> parseLanding(String text){

    HashMap<String, Double> dictionary = new HashMap<String, Double>();
    String s = text.toLowerCase(Locale.ENGLISH).trim();

    s = s.replaceAll("\\s+", "\n");
    s = s.replace("account\nbalance", "__delimiter__");
    s = s.replace("$", "");
    String[] array = s.split("\n");

    boolean enabled = false;
    String accumulate = "";

    for (String item: array){
      String current = item.trim();

      if(current.contains("__delimiter__")){
        enabled = true;
        continue;
      } else if(current.contains("click")){
        break;
      }

      if(enabled){
        try{
          Double a = Double.parseDouble(current);
          dictionary.put(accumulate, a);
          accumulate = "";
        }
        catch (Exception e){
          accumulate += " " + current;
        }
      }
    }
    return dictionary;
}
```