ISSN:0975-9646

Arpita Jadhav Bhatt et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 7 (1) , 2016, 239-248

Disassembling and Patching iOS Applications

Arpita Jadhav Bhatt[#1], Chetna Gupta[*2]

[#*]*Department of Computer Science & IT,*
Jaypee Institute of Information Technology, Noida
A-10, Sector-62, Noida, India

***Abstract*— Apple's iOS is amongst the widely used mobile operating system. Apple follows a mandatory code signing mechanism and an app review for enhancing the level of security for iOS devices. This review procedure ensures that the applications are developed by genuine developers or enterprises. However, recent attacks and data harvesting incidents with the benign applications, have demonstrated that the mandatory code signing procedure is vulnerable to attacks. With the popularity of Smartphone and distribution of third party applications, the malware which is specially designed for the modern mobile platforms is hastily becoming a serious threat. As the users rely more on the third party applications (which span in a wide range of categories like social media and networking, gaming, data management etc.), they put their personal and confidential information at risk. With the widespread use of third party applications, there have been multiple reports of Malware attacks on iPhone. Attackers use these applications to disguise malwares into the user's smartphone. Therefore, for ensuring security of the devices one should perform reverse engineering of mobile applications for catching up the vulnerabilities in them before the attackers do. This paper aims to perform reverse engineering of iOS applications by disassembling, decompiling the application's code using Hopper tool. The paper also demonstrates how an analyst can patch the code in the application for discovering vulnerabilities. In this paper we have demonstrated the process of reverse engineering by disassembling the code with the help of custom application.**

***Keywords*— Reverse Engineering, Disassembling, Decompiling, iOS Applications, Run time Analysis, mobile applications, Run time Modifications**

I. INTRODUCTION

With the popularity of Smartphone, the malware which is specially designed for the modern mobile platforms is increasing rapidly. Further, the problem is multiplexed by the growing convergence of cellular networks, wired and wireless networks, since the developers of the malware or virus can now develop sophisticated malicious software, which is capable of migrating across the network domains [1]. This is done in an effort to exploit the vulnerabilities and the services which are specific with respect to each network. With the growth of Smartphone as well as the networking technologies i.e. the migration from the legacy networks to the converged networks and then to the converged communications has complicated the situation for the providers. They must now deal with millions of mobile devices, which is outside their reach and simultaneously there is huge growth in the wireless network traffic and converged wired traffic [1]. The mobile devices are not equipped with adequate security management capabilities, and they also add up complexity with the massive variety of the applications which are available from their corresponding official or the third-party online application stores [1]. Due to which the network and/or service providers must cope with the management and provisioning of the mobile devices as well as the traffic generated from specific mobile applications, which are traveling over their wireless and wired interfaces. With the continuously growing mobile malware threat, smartphones these days have become the prime target for the attackers as they contain user's personal data. Recent researches have shown that by blending the spyware as a malicious payload with the worms as a delivery mechanism, the malicious applications can be exploited for many facets of identity theft [1].

The developers of the third party application now use polymorphic coding techniques which are capable of affecting multiple platforms simultaneously. By the use of polymorphic coding techniques, developer can try to bypass Apple Store's scrutinizing process or vetting process. Apart from the above, the vulnerable areas include connection with the wireless network, which is insecure for data storage.

Past attacks on iOS Devices

Apple has designed the operating system in such a way that it provides several securities which include: mandatory code signing process, data encryption, creation of binary files etc. [2]. But past attacks which arise due to the presence of third party applications on the iOS devices have discovered different types of vulnerabilities. On a further note there were many vulnerabilities reported in the iOS device architecture which included code signing security bypass vulnerability, bypassing hardware encryption, bypassing keychain encryption etc. Apart from that there have been many data harvesting incidents that included Mogo Road, Storms 8, iSpy, Aurora feint, libtiff, SMS fuzzing etc. [3],[5].The recent attacks include Wirelurker malware attack, masque attack that replaces third party applications with fake applications that have the same name. The infected application masquerades as real application and steals private data [15]. Malwares are dangerous as they are capable of performing actions without user's knowledge, example, making calls charges on phone bill, sending spam messages to the user's contact list. It can also give an intruder remote control over the device. Malware attacks can result in identity theft or financial fraud. In the lieu of the above problems, it is important to examine the behaviour of applications so that data stored in the devices remain intact. Keeping in view the past malware attacks on iOS devices; the aim of the paper is to perform disassembling of iOS application's code for analysing the flow of application and discovering vulnerabilities of an application.

The rest of the paper is organized as follows: Section 2 introduces the process of Reverse Engineering followed by its importance and its process. Section 3 introduces Hopper tool and its features. Section 4 describes how to create an ipa file of application under test and its executable file using Xcode which would be provided as an input for the Hopper tool. Section 5 describes disassembling application's code using Hopper Disassembler. Section 6 presents analysing iOS applications using Hopper, followed by Section 7 that demonstrates Run time modification of application's code by using code patching. Section 8 includes the Conclusion part.

The following section will introduce the importance of reverse engineering, its process and what are the essentials required for performing reverse engineering.

II. Reverse Engineering

The section describes what is reverse engineering and the process of Reverse Engineering. Reverse engineering of applications is useful when the application's source code is not available, so the analysts or users can disassemble the binaries and examine the type of data associated with an application. Cracking mobile applications helps an analyst to trace the application's flow when we are conducting security assessment [6]. Reverse engineering also aids the testers in analysing the work flow and weak points of the application.

A. Process of Reverse Engineering

The process of reverse engineering is a manual as well as an intensive process which requires knowledge of various mobile operating systems and architectures. For performing engineering of applications the testers or the analysts must have in-depth knowledge of the platform, on which the application runs. The various mobile operating systems available are Android, Apple's iOS, Windows, Symbian etc. [6]. Apart from the platforms and software development kit required to create mobile applications, the analysts must be familiar with the ARM CPU [10], [14] architecture that most of the mobile devices use.

When performing reverse engineering, the analysts must overcome several challenges as the developers of the mobile operating systems impose security limitations (Permission Based model while installing applications). The third party applications are signed and run in their own sandboxes which are tied to the user profiles. Each mobile platform uses different configuration files and requires different tools for decompiling and disassembling the code. For Example Android platform uses Android developer tools, SDK manager, and Eclipse plug-in for development, while iOS platform uses Xcode as Integrated Development Environment and Objective-C as programming language.

Apart from the IDE's the executable files are also different for each platform, for Example: files developed for Android devices have .apk extension while the applications developed for iOS devices have .app extension. On a further note, for gaining root level access for iOS devices, the users must first jailbreak their device using Cydia Software and then load the applications which may be required for testing. Hence the process of reverse engineering requires an in-depth knowledge of different mobile platforms, types of executable files generated by each platform, understanding of programming language which is required to develop the applications. The process of reverse engineering of the mobile applications, allows the analyst to get an insight with respect to the application's behaviour and modify them to discover the vulnerable area in an application, before the real intruders do [6]. For performing reverse engineering of iOS applications our paper demonstrates disassembling of code using Hopper tool. The tool helps in static and dynamic analysis of executable files or application's binaries [4]. It is a tool for Linux and OS X and helps in disassembling, debugging and decompiling the executable. It has support for 32/64 bit configuration [7]. The following section introduces Hopper tool followed by subsequent sections that demonstrate how to create an input file for hopper tool, i.e. the ipa file [8] and how to perform run time analysis and modifications using this tool.

III. Introduction To Hopper

Hopper is a reverse engineering tool and is compatible with Mac, Linux and Windows Operating System. In this paper, we have used Hopper V3 on OS X 10.10.4. The important features of this tool are mentioned below [7]:

1. It is native as it can be deployed on various operating systems.
2. Extensible as it allows the user to create own file format.
3. Capable of generating Control flow graphs and Pseudo codes that help an analyst to examine the behaviour of the application under analysis.
4. Support for Objective C language [11]: The tool is specialized in retrieving Objective C information. It can analyse and retrieve sent messages, strings and variables [11].
5. Support for Python Scripts.
6. Compatible with debugger: It can use GDB (Gnutella Debugger) [22] and LLDB [13].

A. Graphical User Interface of Hopper

Figure 1 represents the graphical user interface for hopper [7]. The left portion represented by block 1, consists of labels and strings. It consists of all symbols which are defined in the application's executable file. It also includes a search bar for searching strings inside the application's binary file.

The upper portion (marked as block 2 and block 3 in figure1) is the tool bar which supports various types including **D:** Data (Hopper sets an area to data, type when the bytes represent a constant, array of integers etc.), **A:** ASCII (American Standard Code for Information Interchange and denotes a null terminated string usually a C string), **C:** Codes (can be instructions like JMP, POP, PUSH, MOV etc.), **P:** Procedure (If a part of method is successfully reconstructed by Hopper, then this byte is associated with a procedure), and **U:** Undefined (Portion that is not explored by Hopper). The users can change the type by selecting options from the toolbar [7]. The portion highlighted by block 3 includes the options for generating Pseudo code and Control flow graphs.

The central portion highlighted by block 4 shows the extracted assembly language code. Once the applications binary is processed by Hopper, the assembly level instructions are displayed in this section. The right portion depicted by block 5 represents the Inspector. It displays Instruction encoding format, comments, colour tag and graphical views etc. [7].

Working of Hopper: Hopper transforms the application's binaries, set of bytes into human readable format. It associates a type for each byte of the file and performs automatic analysis of code as soon as we load the application's binary [7]. Hopper provides a user with several options for performing operations such as producing assembly text files, produce a new executable (this option allows the users to patch the code and override the existing code).

The next section demonstrates how users can create an .ipa [8] file which is the input file for the Hopper tool.

IV. Creating An Ipa File And An Application's Executable File Using Xcode

This section presents how to create an .ipa file and application's executable file by using Xcode Simulator.

An .ipa file is an archive file for iOS applications that stores iOS apps. An .ipa file is compressed with binary for ARM architecture [8]. The .ipa file can be installed only on iOS device [8]. The .app file is the application bundle which contains executable file, icon image of the application etc. The ipa file contains .app bundles and files meant for iTunes. The ipa is provided to Hopper as an input for disassembling the application's code and generate assembly language instructions. Once hopper tool has disassembled the code, analysts can then analyse the overall flow of the application, information on the necessary method, get control graph of the application along with Pseudo code. The analyst can further apply code patching technique in which they can modify the assembly language instructions to change the flow of application.

A. Creating an .ipa file using Xcode Simulator

For creating an .ipa file the user first needs to install Xcode under the applications folder [9]. Next step is to open the application folder and select Xcode and right click and select 'Show Package Contents'. In this paper, we have used Xcode version 5.1. For creating an .ipa file, first open Xcode and create a new project or open up an existing project. For our tests, a custom demo application "EmailVerifer" was used. In this the user enters email and password and clicks on sign-in button. If the entered value matches with the stored value in the application, the user is directed to next screen else an alert message is displayed. Apart from that the users can click on Register button, forgot user and forgot password buttons. Figure 2 (next page) depicts the general flow of the application when the user has entered correct email-id and password. The application then directs the user to next screen.

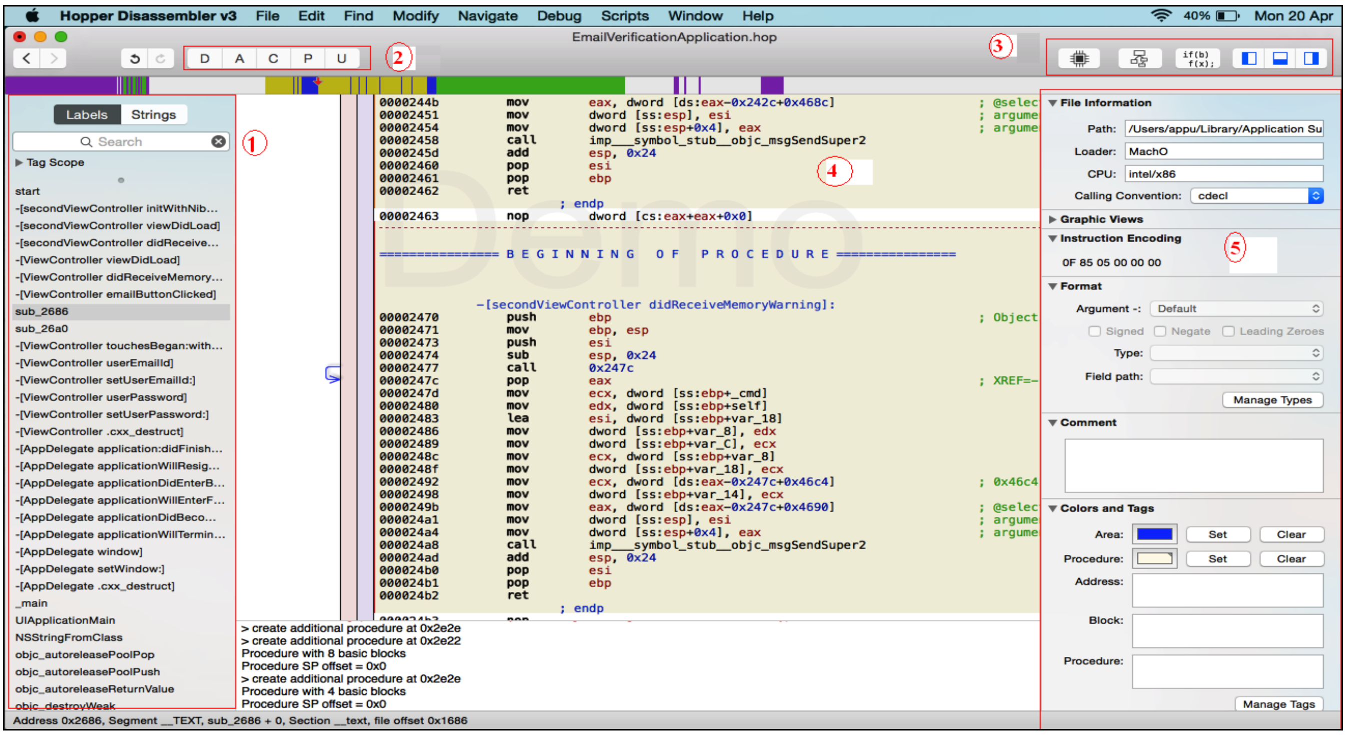

Figure 1. Graphical User Interface for Hopper tool.

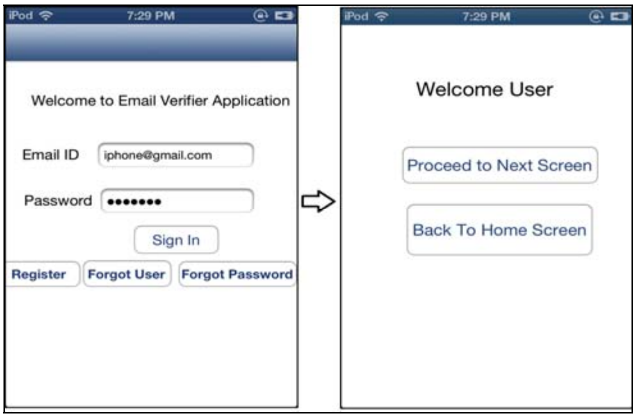

Figure 2. Flow of the "EmailVerifer" application if the user enters correct Email-id and password.

If the user enters an email-Id and password combination that does not match with the password which is stored in the application's code, the user gets an alert message. Figure 3 depicts the flow of application when user has entered incorrect email–Id.

Figure 3. Flow of the "EmailVerifer" application if the user enters incorrect Email-id and password.

After installing the custom demo application, or creating a new application, following steps are necessary for creating archive file and ipa file.

Steps for creating Archive file and .ipa file

1. For generating the .ipa file we need to run the application on the iOS device. After running the application, select Product and click build Archive; this will create the archive for the application under test [9]. Figure 4 depicts how we can create the Archive for iOS application using Xcode.

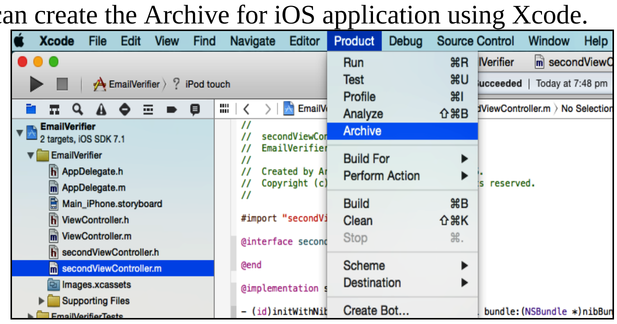

Figure 4. Creating Archive of the application from Xcode.

2. For opening the Archive file after creation (Figure 4) select Window from Xcode and click Organizer then click Archives, the following figure displays the archive file for our project. Click on the archive and right click on select "Show in Finder" [9]. Figure 5 demonstrates the snapshot for the Archive created.

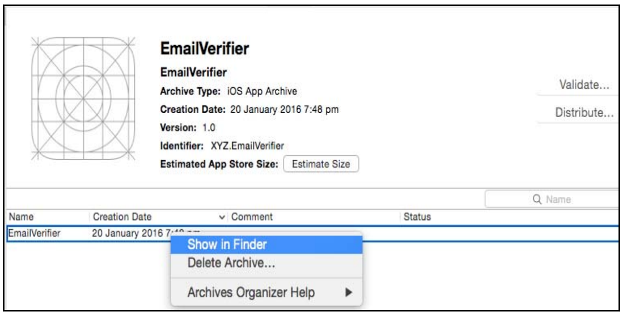

Figure 5. Archive of the application under test using Xcode.

3. As shown in Figure 6, right click the archive file and select "Show Package Contents". Select Products -> Applications-> Archived Project folder -> Archived Project file for locating the .ipa file.

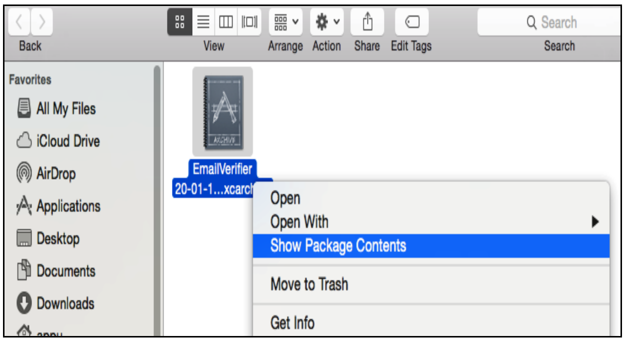

Figure 6. Archive file

4. Figure 7 depicts the .ipa file. This file can be selected for providing input to Hopper disassembler. The analyst can simply drag the .ipa file and into Hopper disassembler.

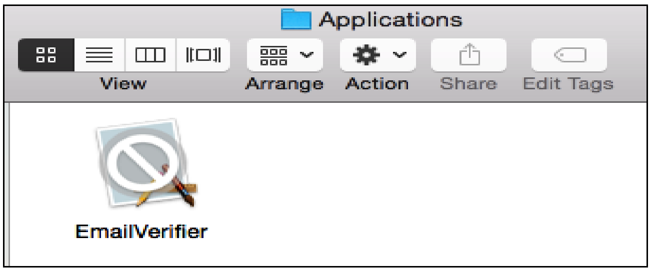

Figure 7. .ipa file

The next section will demonstrate how we can disassemble application's code after providing the .ipa file as an input to Hopper disassembler.

V. Disassembling Application's Code Using Hopper Disassembler

The section covers how an analyst can perform reverse engineering by disassembling the code using Hopper tool [7], analyse the workflow of the application, identify the vulnerable areas. As the process of reverse engineering requires users or analysts to have in-depth knowledge of the platform on which they work, for disassembling the code for iOS applications we require .ipa file of the application that needs to be analysed. After generating the .ipa file, the next step is to provide the .ipa file as an input to the hopper for disassembling the code (Refer section IV). The application which needs to be analysed is EmailVerifer application, the user's email Id and password are hard coded in the application, if the user enters correct Email Id and password combination, the user is validated to next page or else an alert message is displayed. Drag the .ipa file into assembler, select ARM v7 and click on next (Refer figure 9 and 10). An ARM (Advanced Risc Machines Processor) consists of family of Instruction Set of CPU's which are based on reduced instruction set computing (RISC) [10] [14]. ARM processors are widely used in electronic devices such as Tablets, Smartphones, multimedia players etc. ARM-7 family is most extensively used 32-bit embedded processor family [14] [16]. Select ARM v7 option, then click on Next and select MACH-O format (Figure 9) [17] [18] [19].

After selecting the above options we get the disassembled code. Figure 10 depicts the code generated by Hopper disassembler after dragging the .ipa file

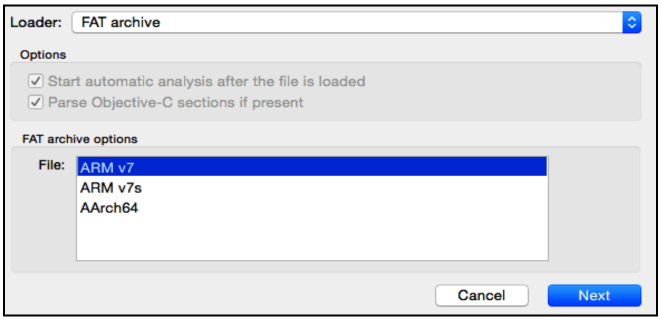

Figure 8. Options for disassembling the .ipa file

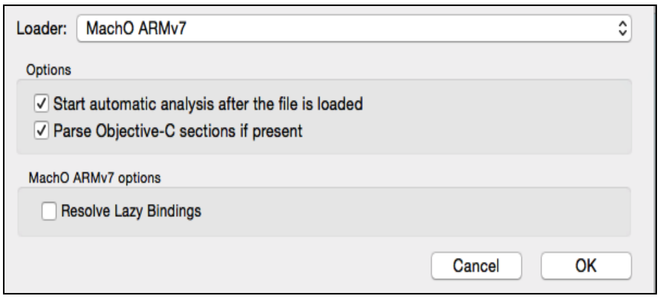

Figure 9. Options for disassembling the .ipa file

.

Figure 10 depicts the code disassembled by Hopper tool which comprises of all necessary methods used while developing the application. It also depicts the necessary labels and methods used inside the application. Figure 11(Next Page) describes the code of the EmailVerifer application (in Xcode) that has Email Id hard coded inside the application. The figure depicts an overview of the application.

Figure 10. Options for disassembling the .ipa file

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

- (BOOL) validateEmailId: (NSString *) user {
    NSString *emailEnter = @"iphone@gmail.com";
    NSPredicate *emailTest = [NSPredicate predicateWithFormat:@"SELF MATCHES %@", emailEnter];
    return [emailTest evaluateWithObject:user];
}

-(IBAction)buttonPressed:(id)sender{

        if([self validateEmailId:[_userEmailId text]] ==1)
    {
        [self performSegueWithIdentifier:@"UserPage" sender:self];
    }
    else{
        UIAlertView *alert = [[UIAlertView alloc] initWithTitle:@"Message" message:@"You have entered Incorrect Email id or
            Password" delegate:self cancelButtonTitle:nil otherButtonTitles:@"OK", nil];
        [alert show];

    }
}
- (IBAction)registerNewUser:(id)sender {
    [self performSegueWithIdentifier:@"RegisterMe" sender:self];

}
- (IBAction)generateUserName:(id)sender {
    [self performSegueWithIdentifier:@"ForgotUserId" sender:self];

}
- (IBAction)generateUserPassword:(id)sender {
    [self performSegueWithIdentifier:@"ForgotUserPassword" sender:self];

}

-(void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    [self.userEmailId resignFirstResponder];
    [self.userPassword resignFirstResponder];
}
```

Figure 11. Code for EmailVerifer Application (Xcode)

```
                -[ViewController buttonPressed:]:
0000a9b0    push    {r4, r5, r6, r7, lr}                 ; Objective C Implementation defined at 0xc134 (instance)
0000a9b2    add     r7, sp, #0xc
0000a9b4    str     r8, [sp, #0xfffffffc]!
0000a9b8    sub     sp, #0x10
0000a9ba    mov     r8, r0
0000a9bc    movw    r0, #0x1f68
0000a9c0    movt    r0, #0x0
0000a9c4    add     r0, pc                               ; objc_ivar_offset_ViewController__userEmailId
0000a9c6    ldr     r0, [r0]                             ; objc_ivar_offset_ViewController__userEmailId
0000a9c8    add     r0, r8                               ; argument #1 for method imp___symbolstub1__objc_loadWeakRetained
0000a9ca    blx     imp___symbolstub1__objc_loadWeakRetained
0000a9ce    mov     r5, r0
0000a9d0    movw    r0, #0x1e10
0000a9d4    movt    r0, #0x0
0000a9d8    add     r0, pc                               ; @selector(text)
0000a9da    ldr     r1, [r0]                             ; @selector(text), argument #2 for method imp___symbolstub1__objc_msgS
0000a9dc    mov     r0, r5                               ; argument #1 for method imp___symbolstub1__objc_msgSend
0000a9de    blx     imp___symbolstub1__objc_msgSend
0000a9e2    mov     r7, r7
0000a9e4    blx     imp___symbolstub1__objc_retainAutoreleasedReturnValue
0000a9e8    mov     r6, r0
0000a9ea    movw    r0, #0x1df8
0000a9ee    movt    r0, #0x0
0000a9f2    mov     r2, r6
0000a9f4    add     r0, pc                               ; @selector(validateEmailId:)
0000a9f6    ldr     r1, [r0]                             ; @selector(validateEmailId:), argument #2 for method imp___symbolstub
0000a9f8    mov     r0, r8                               ; argument #1 for method imp___symbolstub1__objc_msgSend
0000a9fa    blx     imp___symbolstub1__objc_msgSend
0000a9fe    mov     r4, r0
0000aa00    mov     r0, r6                               ; argument #1 for method imp___symbolstub1__objc_release
0000aa02    blx     imp___symbolstub1__objc_release
0000aa06    mov     r0, r5                               ; argument #1 for method imp___symbolstub1__objc_release
0000aa08    blx     imp___symbolstub1__objc_release
0000aa0c    uxtb    r0, r4
0000aa0e    cmp     r0, #0x1
0000aa10    bne     0xaa3a
```

Figure 12. Disassembled code by Hopper when user selects 'buttonpressed' method

From the figure 11 we can identify the 'buttonpressed' method is invoked when the user clicks on sign-in button. The section described how a user can provide an .ipa file as an input to Hopper tool and analyse the disassembled code. The following section will demonstrate the analysis of EmailVerifer application in detail.

VI. Analysis Of An Ios Application Using Hopper

This section will describe how an analyst can inspect the general flow of the application using various features of hopper tool. By using Hopper we can disassemble code, generate a Control flow graph and pseudo code of the application. These help the analyst to get insight with detailed working of the application.

A. *Analysing the Labels and Strings*

The graphical user interface of the tool has a section in the left panel (Refer figure 10) from which we can select the labels and strings used in the application. If we click on labels we can get procedures for any method. Figure 10 also depicts labels for methods such as setUserEmailID, emailButtonClicked, setUserPassword etc. If we click on a label hopper tool will display the procedure of the selected method [7]. Figure 12 depicts the disassembled code if the user selects the 'buttonpressed' method.

Analysts can also analyse the general flow of the application with the help of disassembled code. From figure 13 we can comprehend that the 'buttonpressed' method redirects the user either to the 'User page' or generates an error message. The code is highlighted in figure 13 (Next Page).

```
0000aa16    mov      r3, r8
0000aa18    movt     r0, #0x0
0000aa1c    movw     r2, #0x1e96
0000aa20    add      r0, pc                    ; @selector(performSegueWithIdentifier:sender:)
0000aa22    movt     r2, #0x0
0000aa26    add      r2, pc                    ; @"UserPage"
0000aa28    ldr      r1, [r0]                  ; @selector(performSegueWithIdentifier:sender:)
0000aa2a    mov      r0, r8
0000aa2c    add      sp, #0x10
0000aa2e    ldr      r8, [sp], #0x4
0000aa32    pop.w    {r4, r5, r6, r7, lr}
0000aa36    b.w      0xae74
------------------------------------------------------------------------------------------
0000aa3a    movw     r0, #0x1daa               ; XREF=-[ViewController buttonPressed:]+96
0000aa3e    movt     r0, #0x0
0000aa42    movw     r2, #0x1dc8
0000aa46    movt     r2, #0x0
0000aa4a    add      r0, pc                    ; @selector(alloc)
0000aa4c    add      r2, pc                    ; objc_cls_ref_UIAlertView
0000aa4e    ldr      r1, [r0]                  ; @selector(alloc), argument #2 for method imp___symbolstub1__objc_msg
0000aa50    ldr      r0, [r2]                  ; objc_cls_ref_UIAlertView, argument #1 for method imp___symbolstub1__
0000aa52    blx      imp___symbolstub1__objc_msgSend
0000aa56    movw     r1, #0x1d94
0000aa5a    movs     r5, #0x0
0000aa5c    movt     r1, #0x0
0000aa60    movw     r2, #0x1e5e
0000aa64    add      r1, pc                    ; @selector(initWithTitle:message:delegate:cancelButtonTitle:otherButt
0000aa66    movt     r2, #0x0
0000aa6a    movw     r3, #0x1e62
0000aa6e    add      r2, pc                    ; @"Message"
0000aa70    movt     r3, #0x0
0000aa74    ldr      r1, [r1]                  ; @selector(initWithTitle:message:delegate:cancelButtonTitle:otherButt
0000aa76    movw     r6, #0x1e68
0000aa7a    add      r3, pc                    ; @"You have entered Incorrect Email id or Password"
0000aa7c    movt     r6, #0x0
0000aa80    str.w    r8, [sp]
0000aa84    add      r6, pc                    ; @"OK"
0000aa86    str      r5, [sp, #0x4]
0000aa88    str      r6, [sp, #0x8]
0000aa8a    str      r5, [sp, #0xc]
0000aa8c    blx      imp___symbolstub1__objc_msgSend
```

Figure 13. Analysis of general flow of application

By analysing the labels we can get the method names as well as the procedures corresponding to each method. The other important feature of this tool is that we can examine the values in the Strings. Also the values extracted in the Labels and Strings help user to discover other method names, their procedures used in the application.

B. *Analysing the Pseudo Code*

Another important feature of the tool is that we can generate Pseudo code of the application under analysis. Pseudo code is a notation which resembles a programming language in a simplified form. It is a detailed and readable description of what a program or application must do. We can generate Pseudo code for any method, select the method name from the labels and click the Pseudo code option from the tool bar. Here we have selected the 'buttonpressed' method from the labels and generated the Pseudo code [7]. Figure 14 depicts the Pseudo code for the 'buttonpressed' Method.

```
void -[ViewController buttonPressed:](void * self, void * _cmd, void * arg2) {
    r7 = &arg_C;
    sp = sp + 0xfffffffffffffffc - 0x10;
    r8 = self;
    r5 = objc_loadWeakRetained(*objc_ivar_offset_ViewController__userEmailId + r8);
    r0 = [r5 text];
    r7 = r7;
    r6 = [r0 retain];
    r4 = [r8 validateEmailId:r6];
    [r6 release];
    r0 = [r5 release];
    asm{ uxtb      r0, r4 };
    if (r0 == 0x1) {
            r1 = @selector(performSegueWithIdentifier:sender:);
            r0 = r8;
            Pop();
            Pop();
            Pop();
            Pop();
            Pop();
            r0 = [r0 performSegueWithIdentifier:r2 sender:STK3];
    }
    else {
            r0 = [UIAlertView alloc];
            var_0 = r8;
            arg_4 = 0x0;
            arg_8 = @"OK";
            arg_C = 0x0;
            r4 = [r0 initWithTitle:@"Message" message:@"You have entered Incorrect Email id or
Password" delegate:STK1 cancelButtonTitle:STK0 otherButtonTitles:STK-1];
            [r4 show];
            r0 = r4;
            Pop();
            Pop();
            Pop();
            Pop();
            Pop();
            r0 = [r0 release];
    }
    return;
}
```

Figure 14. Pseudo Code for Email Verification Application

By examining the Pseudo code we can see the comparison of register values which contain the email id and password combination (in form of strings). If the strings entered while running the application match with the hardcoded Email-Id (which is 'iphone@gmail.com'), then the user is redirected to the next screen or else an Alert message is generated. The Pseudo code helps an analyst to get insight into the application's variables and assembly language instructions.

The code highlighted in the Figure 14 depicts the cases when the user has entered correct Email-id in which a user page will be called or if the user has entered incorrect email an alert message would be prompted. Apart from generation of labels, Strings and Pseudo code, one can generate control flow graph with respect to the application.

Following sub-section demonstrates the use of Control Flow Graph for the EmailVerifer application.

C. *Analysing the Control Flow Graphs*

We can generate Control Flow Graphs by selecting the Control Flow graph option from the tool bar. A Control Flow Graph represents the behavior of the application or a program in a graphical format. It describes all the paths which may be traversed during the execution of an application.

Figure 15. Analysis of control flow graph for Email Button Clicked method

Hopper provides option to generate the control flow graph in the form of a PDF file also. Figure 15 depicts the control flow of the application. The analyst can analyze the flow of application at a detailed level. For generating the Control Flow Graph we can select the name of the method from the Labels section and then from the tool bar, we can select the option to generate Control Flow Graph.

Figure 15 is one of the parts of the control flow graph wherein we can observe that incorrect email and password combination entered by user leads to an area on the left hand side, whereas the correct Email Id and Password combination leads to right hand side. The Left hand side portion has the incorrect Email or Password alert message and the right hand side portion provides navigation towards the user page. A blue colour line on the Control flow graph indicates whenever the condition is true. It would then execute the block highlighted by blue colour. If the condition is false then application will execute the block highlighted by red colour.

Our goal would be to redirect to the right hand side and bypass the login method. By bypassing login method, user can thus access the application without actually entering correct credentials. In this way, we can test the application and discover the area which can be exploited by the intruder. Similar scenarios can be exploited in applications which require login id\password or social networking apps where the user is validated, or in gaming applications when the level requires some amount of coins or energy levels before starting it.

Figure 15 depicts the control flow graph of the application when 'buttonpressed' method is called.

D. Analysing the ARM Assembly language instructions

The iOS devices are based on the ARM architecture. The language used for creating iOS application is Objective-C language. The Objective-C language is object oriented language which provides the users with a dynamic runtime environment [23] [11]. The Objective-C code that is used for developing iOS application is converted into ARM assembly language and then to machine code [22]. If an analyst has a good understanding of the assembly language, it is possible to decipher the code which is written in Objective-C during runtime and modify it [22]. From Figures 10, 12, 13, 14 and 15 we can trace out the assembly language instructions generated by the Hopper tool. The instructions are BNE (Not Equal) [20] [24], mov (Move) [24], ADD, PUSH and other test instructions. The assembly language comprises of **Instructions**: which is a statement executed at runtime, **Operands**: the entities operated by the instructions and **Addresses**: the locations in memory to store data [21].

In figure 15, we have an instruction 'bne 0xaa3a', a block which corresponds to right hand side where the control of the application reaches after the user is validated. So our goal will be to modify the assembly language instruction such that the flow is always directed towards the right hand side (Described in Section 7).

The section included how can an analyst perform analysis of iOS applications by analysing the assembly code, check for the methods and procedures used in an application, generate Pseudo code and examine the flow of the application by using Control flow Graph. Next section will describe how an analyst can perform Run time modification by patching the code inside the application and produce a new executable file.

VII. PERFORMING RUN TIME MODIFICATION IN IOS APPLICATION BY CODE PATCHING

This section will describe how an analyst can modify application's behaviour by code patching. Code patching allows modifying an instruction so that application's behaviour or flow can be altered.

From above section we observed that a branch instruction is invoked (Refer Figure 12, 15) when user enters an Email-Id and Password. In order to alter application's behaviour such that it always invokes the user page irrespective of email id (Refer Figure 2), Click on Modify instruction and select Assemble instruction [7] [12] (Refer figure 16).

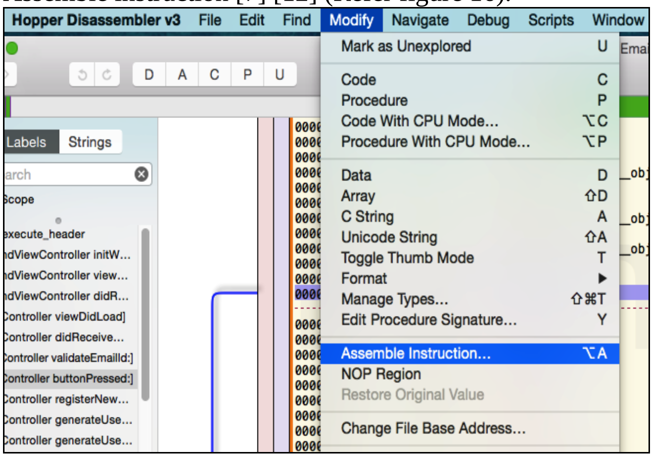

Figure 16. Modifying the assembly instructions

Select the bne (Not Equal) instruction (analysed from the control flow graph, Figure 15) and type *b 0xaa3a* [7] [24].

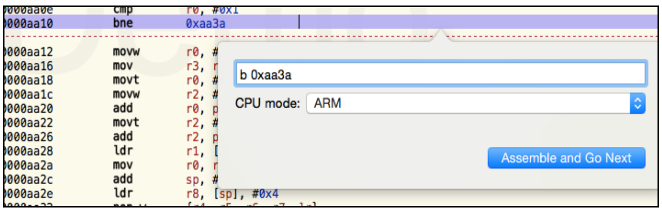

Figure 17. Modifying the assembly instructions for 'buttonpressed' clicked method

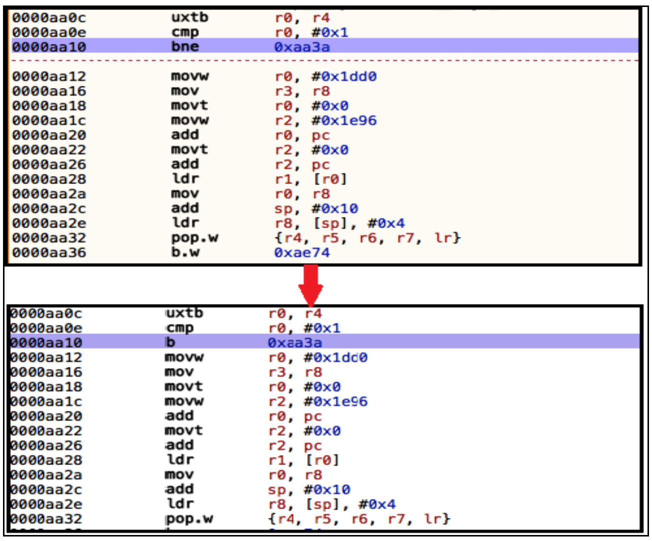

Figure 18. After Modifying the assembly instructions for 'buttonpressed' clicked method

Figure 18 depicts the before and after modification of the assembly instruction *bne* to *b* [24].

Now, to save the code, Click on the file option from the toolbar and select Produce New Executable, this will override the existing code and patch the new code. It is important to save the file [7] [8] after patching the code. An alternative approach for modifying the behaviour of the application is to use NOP Region option. Figure 19 depicts the options a user can select to modify the *bne* instruction [24].

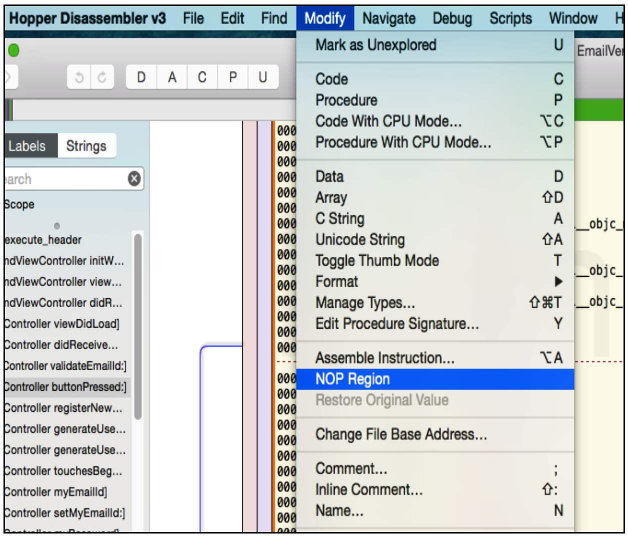

Figure 19. NOP region to modify the assembly code.

Select the *bne* (Not Equal) [24] instruction (Figure 15) and select NOP region [7] (Refer figure 18). Figure 20 depicts the before and after modification of *bne* instruction [24].

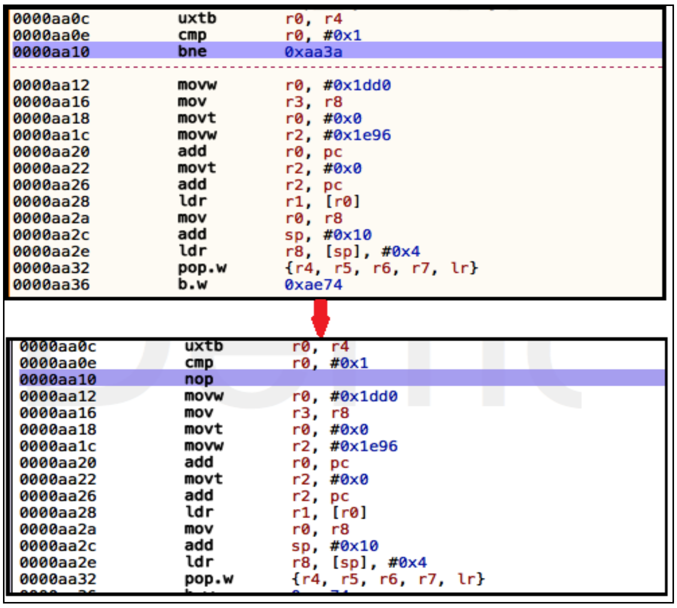

Figure 20. After modification

The above steps will patch the binary code and every time when user runs the EmailVerifier application, the user will be automatically redirected to the 'User Page' even with incorrect Email Id and Password combination.

VIII. CONCLUSION

The paper presents how we can perform reverse engineering of iOS applications by disassembling the code using Hopper tool. For performing reverse engineering, one of the methods used was disassembling application's code. After analysing the disassembled code, it was observed that by patching the instructions by using code patching technique, we were able to modify the applications behaviour. Similar method, if applied to other applications, would expose vulnerable areas which the attacker may exploit in order to bypass the logic of the application code. The demonstration of demo application is supported with the help of code snippets and figures. With the help of this tool we have extracted the assembly language instructions of the application under analysis. The code patching technique demonstrated in Section 7 can be used to bypass certain methods in an application and hence examine probable vulnerabilities. The suggested work thus helps the Smartphone users to study the behaviour of installed applications, enhance security and in turn protect their privacy.

REFERENCES

[1] Dimitrios Damopoulos, Georgios Kambourakis, Stefanos Gritzalis and Sang Oh Park, 'Exposing mobile malware from the inside (or what is your mobile app really doing?)', Peer-to-Peer networking and Applications, 29th October 2012, Print ISSN: 1936-6442, Publisher Springer US

[2] Tim Werthmann, Ralf Hund, Lucas Davi, Ahmad-Reza Sadeghi and Thorsten Holz (May 2013), "PSiOS: Bring Your Own Privacy & Security to iOS Devices", 8th ACM Symposium on Information, Computer and Communications Security (ASIACCS 2013)

[3] Oliver Karow, Symantec Germany GmbH, "Apple iOS Security in the Enterprise". Link:http://www.oliverkarow.de/cruft/Apple%20iOS%20Security%20in%20the%20Enterprise%20WP.pdf

[4] Aswathy Dinesh and Ming Chow, "An analysis of mobile malware and detection techniques"

[5] Charlie Miller , Dion Blazakis, Dino DaiZovi ,Stefan Esser, Vincenzo Iozzo , Ralf-Philip Weinmann "iOS Hacker's Handbook" Publication Date: May 8, 2012 , ISBN-10: 1118204123 | ISBN-13: 978-1118204122 , First Edition, Published by John Wiley & Sons

[6] Megan Horner,' How to Find Vulnerabilities in Mobile Apps through Reverse Engineering', IT Security, January 24,2013, Link: https://dalewifisec.wordpress.com/2013/01/24/how-to-find-vulnerabilities-in-mobile-apps-through-reverse-engineering/

[7] Tutorial Hopper Link: http://www.hopperapp.com/tutorial.html

[8] .ipa (file extension), Wikipedia Link: http://en.wikipedia.org/wiki/.ipa_%28file_extension%29

[9] Majinsmash Gaming ,'How to Make An IPA For Your App In Xcode Without Dev. Account ' Link: https://www.youtube.com/watch?v=aZ747B6MAOA

[10] ARM Processor, Link: http://whatis.techtarget.com/definition/ARM-processor

[11] Stephen G. Kochan,"Programming in Objective-C (Developer's Library): Sixth Edition, Publication Date: December 2013, ISBN-13:978-0-321-96760-2 | ISBN-10:0-321-96760-7, Published by Pearson Education, Inc.

[12] Prateek Gianchandani, "IOS Application Security Part 28 – Patching IOS Application with Hopper". InfoSec Institute, Link : http://resources.infosecinstitute.com/ios-application-security-part-28-patching-ios-application-hopper/

[13] Bart Cone, Hopper + lldb for iOS Developers : A Gentle Introduction Link : http://www.bartcone.com/new-blog/2014/11/26/hopper-lldb-for-ios-developers-a-gentle-introduction

[14] ARM, The Architecture of digital world Link: http://www.arm.com/products/processors/classic/arm7/

[15] Malarie Gokey, Digital Trends" Researchers find new risk in iOS 'Masque Attack' bug", Link : http://www.digitaltrends.com/mobile/ios-bug-masque-attack-news/

[16] William Hohl,"ARM Assembly Language: Fundamentals and Techniques 1st Edition", ISBN-13: 978-1439806104,ISBN-10: 1439806101, Published by Taylor & Francis Group

[17] Wikipedia "Mach - O" , Link: http://en.wikipedia.org/wiki/Mach-O

[18] Sami Samhuri, Basics of Mach-O format, Link:https://samhuri.net/posts/2010/01/basics-of-the-mach-o-file-format/, January 2010

[19] Mac Developer Library OS X ABI Mach-O File Format Reference Link:https://developer.apple.com/library/mac/documentation/DeveloperTools/Conceptual/MachORuntime/index.html#//apple_ref/doc/uid/TP40000895-CH248-SW3

[20] Steve Friedl, "Unixwiz.net Tech Tips Intel x86 JUMP quick reference " Link: http://www.unixwiz.net/techtips/x86-jumps.html

[21] Oracle,'x86 Assembly Language Reference Manual' Link:http://docs.oracle.com/cd/E19120-01/open.solaris/817-5477/ennby/index.html

[22] Arnold Robbins,"GDB Pocket Reference Pocket Reference (O'Reilly)" First Edition Publication Date: May 2005, ISBN-13: 978-0596100278, ISBN-10: 0596100272, Published by O'Reilly Media, Inc.

[23] About Objective C Internet https://developer.apple.com/library/mac/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html

[24] Mark McDermott,"The ARM Instruction Set Architecture", Link:users.ece.utexas.edu/~valvano/EE345M/Arm_EE382N_4.pdf.