



MOBILE APPLICATION SECURITY WITH OPEN-SOURCE TOOLS

Sakthivel Rajendran

Manager - Information Security Risk
Management
Philips India Limited

EMC²

Table of Contents

Continued importance of Application Security	4
Year of major security breaches	4
Instances of web-application security issues which lead to breaches	4
Mobile Application security issues	5
Lessons from security breaches in last year	7
How to incorporate security in development?	9
Building security in Development	9
Initiating SDL.....	10
Requirement gathering:.....	10
Decentralized security ownership in application development	11
Application classification.....	11
Design.....	13
Avoiding top 10 design flaws with IEEE guide	13
13 design principles to ensure application security	15
Threat modeling and Architecture Review Analysis	16
Coding	16
OWASP Mobile Security Project.....	17
ENISA Smartphone Secure Development Guidelines for App Developers	17
Android Security Tips	18
iOS Secure Coding guidelines.....	19
Static Analysis.....	19
Developer training	20
Testing	21
Setting up a Mobile Testing lab	21
Establishing Minimum baseline of Security Test cases.....	22
Security objectives to consider when establishing test cases	24

iOS Security Test Cases	25
Android Security Test Cases	28
Remediation of security issues	30
Maintenance	30
Appendix I: Steps to execute some of the iOS test cases	31
Test Case ID: IOS-01	31
Test Case ID: IOS-02	37
Test Case ID: IOS-05	39
Test Case ID: IOS-08	43
Test Case ID: IOS-09	44
Test Case ID: IOS-10	45
Test Case ID: IOS-11	46
Test Case ID: IOS-12	46
Test Case ID: IOS-19	47
Test Case ID: IOS-22	50
References	53

Disclaimer: The views, processes or methodologies published in this article are those of the author. They do not necessarily reflect EMC Corporation's views, processes or methodologies.

Continued importance of Application Security

Application Software is one of the important components which fuel the so-called third-platform which is congregation of Social, Mobile, Analytics with Big Data, and Cloud computing (SMAC) technology.

Year of major security breaches

Last year -2014, we have seen some of the major information security breaches. It all started with Target retail store security compromise. Around 110 million credit card information of users was stolen. Hackers breached Target's network with stolen credential and installed malware in Point-of-Sale (PoS) system. Flood of high-profile attacks involving credit card information, and personal information were witnessed throughout the year in different organizations around the globe.

PoS malware based attacks were very prominent in the aforementioned security breaches. At least 20 major security incidents affected around 547 million users (source: <http://www.bankinfosecurity.in/infographic-2014s-top-breaches-so-far-a-7408>).

Insecure applications deployed in production systems are one of the most overlooked aspects, which have contributed to some of the compromises.

Instances of web-application security issues which lead to breaches

Orange – a French telecom mobile broadband company – was hacked twice last year. The suspicion was that an injection could have been the modus operandi. SQL injection – wherein a malicious user inserts SQL command in user input entry form to pull out the information from the database. Though SQL injection is a common security issue in web application, it is often overlooked.

A bug in the OpenSSL software - which is designed to encrypt the communication between user's computer and web server, allows anyone to read the memory of the servers protected by OpenSSL. This vulnerability compromises the secret key used to encrypt the communication, username and password, and the actual content being transmitted.

In case of Community Health Systems (CHS) – a health care service provider in US – attackers were able to retrieve user credentials from a Juniper security device via the Heartbleed vulnerability. They then accessed the Virtual Private Network (VPN) of CHS with such credentials and stole approximately 4.5 million patient records. Mumsnet – a community service

platform for mothers – was also a victim of Heartbleed vulnerability in their web application. This resulted in compromise of estimated 1.5 million users' personal information.

Interestingly, goodhearted security researchers notifying the respective organizations to fix the vulnerabilities instead of exploiting the affected companies were witnessed as well.

Chinese retail major Alibaba's e-commerce platform contained security vulnerabilities which could have allowed the hackers to modify the price and shipping address, and have it delivered to their home. Two Israeli researchers representing the security firms AppSec Labs and Cybermoon promptly notified Alibaba and saved it from a potential security incident.

A German security researcher found security vulnerability in Amazon's Kindle Library which could have compromised the entire Amazon.com account. The vulnerability – Cross-Site Scripting (known as XSS) – allowed access to Amazon account cookies and transferred them to the attacker. This could result in compromise of victim's Amazon account.

Online retailer eBay was hit by XSS attack, which resulted in users being redirected to a spoof site set up to steal credentials. Sadly, this was the second security breach in one year for eBay. XSS is a known attack vector. But, often this is overlooked when building an application.

In most cases, the data is stolen with criminal intent. Credit card information, including pin numbers, can be used to do online purchasing. Similarly, when the attacker steals the account information of individuals, they might attempt "spear phishing attack" – which are highly personalized emails designed to trick even security-aware users. The mail would ask the user to click on a link and download malware or key logger software to their system if they do click.

Mobile Application security issues

The situation is even worse in case of mobile applications – wherein the application is downloaded from App store (or other sources) in the user's device itself. The installed application if not protected appropriately can be reverse engineered to get the source code. Vital information like understanding the logic, API used, and internal URLs, can be easily found by the attacker.

Snapchat is an app for mobile-based conversations and chat. Developers of the app claimed that it allows users to send photos and videos which are self-destructing from the company's server and device when the recipient views the snaps. Snapchat app would allow users to share secrets which are not stored anywhere.

However, researchers proved the company's claims were inaccurate. Security vulnerabilities found in the app could easily compromise users' privacy. Static authentication tokens were used to authenticate the user. Symmetric encryption key used for encrypting the snaps were found in the application code itself. Also, the key was the same for both iOS and Android.

A security researcher found a weakness in Starbuck Mobile app. The vulnerability could allow the hacker to gain access to username, email address, and password should the device be stolen. These elements once recovered from an affected user can be used by the malicious user in their own device or online portal of Starbuck to make payments. However, the consolation is that vulnerability existed in iPhone app alone. The issue found was that Starbuck App stores the elements such username and password in clear-text in the user device itself, instead of authenticating user for every new session.

Assessment by another security researcher highlights how individual's privacy is violated. Uber Android mobile app asks for permissions like any other app when installing in the user device. Many of the permissions are legitimate, and in fact enhance rider's experience when availing cab services from Uber.

However, the extent of data collected by the app seems to exceed permissions asked for when installing the app. Uber responded in an official statement that "Our code lists several features that our mobile security vendor offers, but that we do not use". This demonstrates another important aspect – when developing an app, third-party libraries and codes are used without even validating it, resulting in security vulnerabilities which is available for all to exploit.

Research by two security experts representing different security firms reveals that the mobile banking apps of top influential banks around the world has a lot of common security vulnerabilities. These apps were leaking information through insecure coding. For example, these apps are vulnerable to Man-in-the-Middle (MitM) attack, XSS attacks, leakage of sensitive information through system logs, hard-coded credentials in the code, and using non-SSL (HTTP) protocol for transmitting sensitive information between remote server and user device.

HP Mobile application security study released in 2013 reports that more than 2000 mobile applications it analyzed through its SaaS-based scanning services for 601 companies located in 50 countries have exhibited the following security weaknesses:

1. Ignoring Privacy issues
2. Lack of Binary protection
3. Insecure Data storage
4. Transport security
5. Weak server-side controls

Yet another study from HP on Internet of Things (IoT) in 2014 titled, “Internet of Things Research Study” comprising smart web-enabled devices, such as televisions, webcams, home thermostats, door locks, home alarms, etc., reveals the following security weaknesses:

1. Privacy concerns
2. Insufficient authentication and authorization
3. Lack of transport encryption
4. Insecure web interface
5. Insecure software and firmware

Lessons from security breaches in last year

The above incidents and security breaches are only tip of the iceberg. There is no best defense against cyber security risks. We have to assume that the organization is under attack always, and continue to invest in securing the organization IT infrastructure and confidential information from attacks. While it may be difficult to defeat the hackers from breaching the organization's network, we can certainly make it hard for them to do so.

If we take a close look at underlying reasons for some of the breaches and study report from security researchers, we can conclude that security of the application deployed is of paramount importance like the perimeter and gateway level security. Applications exposed on the Internet, and mobile applications installed in the devices are entry points to the organizations network. Insecurity in application can compromise the best of security arrangements.

2013 Global Information Security Workforce Study (GISWS) based on the survey conducted by ISC2 express a couple of aspects clearly, which are important in the context of securing organizations from security breaches. Below are the two aspects from the GISWS report

- Survey respondents ranked Application vulnerabilities as the highest security concern
- Importance of secure software development was rated above the software and hardware solutions in securing the organization's infrastructure

We can expect that application security will continue to be an important area from security professionals' perspective in securing and enabling business operations when the 2015 GISWS report is released.

Over the past decade, companies have taken software security seriously, and tried to build security into the code. However, we still have a long way to go in this area. Recent security incidents brought into visibility that a majority of organizations do not write the majority of the software on their own. Rather, they take the commercial components and open-source components and deliver the business functionality. It is no surprise that when Heartbleed and Shellshock vulnerabilities – which are open-source components – became public knowledge, it affected many companies.

Another stark reality is that penetration tests and regulatory compliance is not a panacea for all the problems. No one can deny the fact the organizations which have experienced security breaches have conducted penetrations testing on their infrastructure/web-facing applications, and subjected their operating environment to many regulatory compliance audits. Penetration testers often operate under the constraint – which actual hackers are not.

Now we're in the midst of third-platform computing evolution. Third-platform technology congregates Social, Mobile, Analytics with Big Data, and Cloud computing (SMAC) technology. It changes the way business and people are using technology. Organizations are embracing third-platform technology to enable corporate information access to employees at any time, from anywhere in the world, using any device.

Application Software is one of the important components which enable social interaction, mobilizing the information access, performing data analytics, and pervasive access thanks to cloud. The smart devices are either embedded with application software on them or allow users

of such devices to install software to make them function for accomplishing an intended objective.

As such, applications are vital in the scheme of things. Securing them from security vulnerabilities and risks are fundamental. Mobile and embedded applications are very similar to web application, but a major difference is that the compiled application is loaded in the device itself, making the probability of attacks high.

The reminder of this article focuses on secure development practices when developing mobile applications. Also, the uses of open-source security tools in securing the application are examined. Exploring free security tools is an option when the resources are hard to get for security programs.

How to incorporate security in development?

Performing security test and incorporating security in the application just prior to release of the software for general users is not an ideal approach. First, remediating the security vulnerabilities in the later stages of Software Development Lifecycle (SDLC) is time consuming and very costly. Second, bolting on security when the actual development is complete ignores the business's requirement of protecting the confidentiality, integrity, and availability of data.

Security best practices, sage advice of security experts, and lessons from the security breaches all suggest one common thing. Build security in the application being developed from the very beginning, rather than bolting it on at the final stage of SDLC. Irrespective of the development methodology followed, be it traditional waterfall, iterative model, Agile/Scrum, the approach of continuously integrating security shall continue to remain the same.

Building security in Development

Secure Development Lifecycle (SDL) aims to incorporate security in all phases of software development activities starting from Requirement Gathering to Testing, Release, and Maintenance. Instead of diving deep into the subject of SDL and how to implement such a program in an organization, we can focus on continuous integration of security with open-source security assessment tools and design principles in securing mobile applications. For a more detailed study of SDL, readers might want to refer the following resources:

1. [Microsoft Security Development Lifecycle](#)
2. [Building Security in Maturity Model \(BSIMM\)](#)

Security in Development



Initiating SDL

Organizations intending to develop secure software application must begin by forming a core group consisting of individuals from development, testing, architecture, security, and project managers to drive the initiative. This group is responsible for embedding security controls from the inception of software development till it is released for production use. Information security control recommendations from this group must be built into the application in all phases of SDLC as appropriate.

Requirement gathering

Embedding security in application development begins at the requirement gathering phase. Apart from business functionality requirements of the software, determine the

1. Tolerance level of the business/customer or whoever would be the user of the application.
2. User-specific security requirements expected in the application. This can include Confidentiality, Integrity, Availability, and Authentication.
3. Importance of the data handled by the application and security requirements to protect it.
4. Compliance and regulatory mandates applicable for the users, region from where the application would be used, or the information handled by the application.
5. Use and Misuse cases from a security perspective.
6. Requirement Traceability Matrix to map requirements with security risks.

All applications being developed may not require stringent security controls. For example, the security consideration for a revenue generating customer facing e-commerce mobile application is much more stringent than the mobile-enabled public web-site of an organization. Also, IT organizations are competing for manpower and resources. Hence, expecting all software development projects to adhere to strict secure development lifecycle may not always be possible.

Establishing an application tier and mapping with security treatment can help both the development and security teams. Many times the development teams give an impression that security testing and remediation consumes enormous duration. Application tier and security treatment coupled with decentralized ownership for security in software development will alleviate the criticism.

Decentralized security ownership in application development

Decentralized security ownership in application development truly takes place when developers are empowered to perform tasks which improve the security posture of the application. It includes imparting knowledge to securely architect the design (to avoid the flaws in the construct of the application), adhere to secure coding guidelines, perform static analysis (to find and remediate code level bugs), and execute dynamic security analysis (to get an attacker view of the running application and remediate them before application is released for users).

Subsequent paragraphs aim to provide guidance on empowering the development team to own security. However, it is very important to remember that security teams would review the security tasks performed by development teams for security assurance.

Essentially, information security teams create a partnership with the development group to secure the organizational information. In this age, where cloud and mobility is breaking the traditional way that IT operates, information security is a joint responsibility.

Application classification

Application classification by making tiers and mapping security considerations for each category is expected to serve the application development and security teams in their mission. Classifying the application based on pre-determined parameters makes the task of application security a shared responsibility.

Below are parameters to create an application-tier:

- Application type
- Developer of the application
- Users of the application
- Hosting environment from where application is made available to users
- Data classification per Organization standards
- Applicability of compliance requirements for the information handled in the application

Sample Mobile Application Assessment Matrix

App Classification	Application Developer	User	Hosting Environment	Data Classification	Compliance Requirement Applicability to business
Mission Critical	Internal Development	Employee managed device	Internal or Private Cloud	Public – Read only	Yes
Business Critical	External Developer from third-party service provider	Partners/External Parties	DMZ	Public – Modify	No
Business Supporting			External Cloud	Confidential	
				Confidential - PII	
				Restricted: Confidential	
Requires Static, Run-Time, and Manual Testing Risk Analysis [OWNED BY DEVELOPMENT+SECURITY TEAM]					
Requires normal SDLC Static and Run-Time Risk Analysis [OWNED BY DEVELOPMENT TEAM, AND REVIEWED BY SECURITY]					

This model works on the assumption that the development team owns and incorporates security tasks when developing software with the supervision/guidance of the security team. For example, development team owning security when *business critical application, handling public read-only data* which is hosted *internally within the organization*. The key here is empowerment of development teams to perform static analysis and dynamic analysis.

Security team involves and owns the security responsibility for the developed application when it is meeting certain criteria, for example – *mission critical application hosted in DMZ environment*. This adds another layer of assurance in developing secure software.

Design

The Design phase of the application where the requirements for functionalities is converted to architecture is an important point to embed security controls for application security.

Constructing secure design will practically minimize the majority of security issues because code level issues can be identified with static analysis or manual code review. On the other hand, design inconsistencies are not visible unless efforts are made to do a threat-model and architecture review. Following secure design principle greatly aids towards this cause.

Avoiding top 10 design flaws with IEEE guide

A common misconception with the developer and security communities alike is that application security is all about finding as many security bugs/findings as possible in a given application. But we're really missing the point and focusing on a wrong area. Instead of addressing the root cause of the problem, often the focus is on a short-term fix for all the security issues in the application.

We often assume that insecurity in software arises due to software errors. While this is true, it may not be the case always.

Bugs in software refer to the software coding error, whereas flaws are defects in the business logic and the way in which the software is built to operate. In fact, more than half of the issues encountered in software security are associated with design flaws that become evident later.

Experts suggest the best way to arrest the failing software is strenuous efforts to mitigate the design flaws.

Recognizing the importance of design in security of the applications, Institute of Electrical and Electronics Engineers (IEEE) launched the Center for Secure Design (CSD) initiative. CSD immediately got into action and issued Top 10 design flaws and ways to avoid them.

S.No	Design Flaw	Recommendation to consider when designing an application solution
1	Incorrect trust assumptions	Earn or give, but never assume, trust
2	Broken authentication mechanisms that can be bypassed or tampered with	Use an authentication mechanism that cannot be bypassed or tampered with
3	Neglecting to authorize after authentication	Authorize after you authenticate
4	Lack of strict separation between data and control instructions, and as a result processing control instructions received from an untrusted source	Strictly separate data and control instructions, and never process control instructions received from untrusted sources
5	Not explicitly validating all data	Define an approach that ensures all data are explicitly validated
6	Misuse of cryptography	Use cryptography correctly
7	Failure to identify sensitive data and how they should be handled	Identify sensitive data and how they should be handled
8	Failure to consider the users	Always consider the users
9	Misunderstanding how integrating external components change an attack surface	Understand how integrating external components changes your attack surface
10	Brittleness in the face of future changes made to objects and actors	Be flexible when considering future changes to objects and actors

13 design principles to ensure application security

Security expert Gary McGraw - CTO of software security consulting firm Cigital, and author of many best-selling security books - suggests 13 design principles in order to protect computer systems. Gary's recommendation is based on the paper titled "The Protection of Information in Computer Systems" published by Jerry Saltzer and Michael Schroeder in 1975. What they said four decades ago is valid even today.

Thirteen design security principles are

1. **Secure the weakest link:** When it comes to secure design, make sure to consider the weakest link in your system and ensure that it is secure enough.
2. **Defend in depth:** If you are designing an application, prevent single points of failure with security redundancies and layers of defense.
3. **Fail Securely:** Design your system to fail securely – meaning restore the settings to default or known better state instead of allowing any vector for attack.
4. **Grant least privilege:** When you do have to grant permission for a user or a process to do something, grant as little permission as possible.
5. **Separate privileges:** When attackers are able to bypass one privilege but not a second, they may not be able to launch a successful attack. Keep privilege sets apart.
6. **Economize mechanism:** Complexity in design increases the security risk. The best way to minimize the security risk is by lessening complexity of the design.
7. **Do not share mechanism:** By not sharing objects and access mechanisms between users, you will lessen the possibility of security failure.
8. **Be reluctant to trust:** Assume that the environment where your system operates is hostile. Don't let just anyone call your API, and certainly don't let just anyone gain access to your secrets! If you rely on a cloud component, put in some checks to make sure that it has not been spoofed or otherwise compromised.
9. **Assume your secrets are not safe:** Security is not obscurity, especially when it comes to secrets stored in your code. Assume that an attacker will find out about as much about your system as a power user, maybe more.
10. **Mediate completely:** Every access and every object should be checked, every time.
11. **Make security usable:** If your security mechanisms are too odious, your users will go to great length to circumvent or avoid them. Make sure that your security system is as secure as it needs to be, but no more

12. **Promote privacy:** When you design a system, think about the privacy of its ultimate users
13. **Use your resources:** If you're not sure whether your system design is secure, ask for help. Architectural risk analysis is hard, but there are people who have been doing it well for decades.

Threat modeling and Architecture Review Analysis

Performing threat model and Architecture Risk Analysis of the design gives the measure of how likely the software will be attacked and the extent of damage it could cause. It can be started by building a high-level overview of the proposed system. Then subject the design to an analysis from an attacker's perspective – i.e. finding ways to exploit the application.

Coding

During the coding phase, business/customer/product requirements are converted into an application. The input for this comes from the previous phases in SDLC – Requirement gathering, and Design. Developers convert the design documents into functioning software. Incorrect writing of a code results in software error. Coding errors can be reduced greatly when secure coding guidelines are applied in application development.

Coding guidelines can be either (a) generic, which is applied in all development environments irrespective of the platform chosen to construct an application. Open Web Application Security Project (OWASP) and European Union Agency for Network and Information Security (ENISA) secure mobile application guidelines are generic guidelines, or (b) platform specific – coding guidelines related to a development platform like Android or iOS.

Another important consideration is use of development frameworks and third-party libraries. Some security breaches have happened due to the vulnerabilities found in them, examples include OpenSSL flaw that lead to Heartbleed vulnerability.

It is recommended to create an inventory of open-source and third-party libraries used in the application which is being developed, and retain the same as part of the development artifacts. A benefit of doing this is that when any security incident takes place involving these libraries, remediation would be very quick.

Another advantage is proactive monitoring of vulnerabilities in the open-source components by referring to the inventory sheet, and taking appropriate corrective action when something undesirable is forthcoming.

It is also worthwhile to vet the open-source and third-party libraries. The objective is to minimize vulnerabilities like backdoors embedded in them or other security issues. Free security scanning services provided by Coverity for open-source projects - <https://scan.coverity.com/> - can be leveraged for this purpose.

OWASP Mobile Security Project

Open Web Application Security Project (OWASP) is a security community dedicated to application security. OWASP creates freely available security resources – software development best practices, methodologies, and security tools. One such community resource is OWASP Mobile security project, which includes top 10 risks for mobile applications, security controls to mitigate such risks, security tools, mobile security testing guide, and guidance for mobile application threat model.

It is valuable to leverage these resources when designing a mobile application.



ENISA Smartphone Secure Development Guidelines for App Developers

Another valuable guidance worth considering when developing a mobile application is from European Union Agency for Network and Information Security (ENISA). It was developed jointly

with OWASP mobile security project. It is written for developers of mobile applications as a guide to developing secure apps.

Below are the 10 mobile security controls from ENISA.

1. Identify and protect sensitive data on the mobile device.
2. Handle password credentials securely on the device.
3. Ensure sensitive data is protected in transit.
4. Implement user authentication and authorization and session management correctly.
5. Keep the backend APIs (services) and the platform (server) secure.
6. Secure data integration with third party services and applications.
7. Pay specific attention to the collection and storage of consent for the collection and use of user's data.
8. Implement controls to prevent unauthorized access to paid-for resources (wallet, SMS, phone calls, etc.).
9. Ensure secure distribution/provisioning of mobile applications.
10. Carefully check any runtime interpretation of code for errors.

Android Security Tips

The Security Tips section in the Android developer portal details the security best practices for developing an Android Application. The following areas are described in detail for the developer to carefully consider and implement when developing an Android mobile application.

1. Storing Data
2. Using permissions
3. Using Networking
4. Performing input validation
5. Handling user data
6. Using webview
7. Using Cryptography
8. Using Interprocess communication
9. Dynamically loading code
10. Security in Virtual Machine
11. Security in Native code

More details can be found in the URL - <http://developer.android.com/training/articles/security-tips.html>

iOS Secure Coding guidelines

Apple's secure coding guidelines provide invaluable resources to developers for building an iOS application which can minimize security attack. This guideline details steps to safeguard from the following vulnerabilities.

1. Buffer overflows
2. Unvalidated input
3. Race Conditions
4. Interprocess communication
5. Insecure File operations
6. Access control problems
7. Secure Storage and encryption
8. Social engineering

This guideline also contains Security Development checklist and Third-party software security guidelines. More details can be found in

<https://developer.apple.com/library/ios/documentation/Security/Conceptual/SecureCodingGuide/SecureCodingGuide.pdf>

Static Analysis

Running static analysis on the source code early in the lifecycle helps to fix code level bugs before the application is released for general use. Static analysis finds incorrect coding that can potentially cause security risks. Analysis is performed without actually executing the program. Entire source code or binary is covered in this kind of analysis. It can be built in the development process, and performed early in software development lifecycle.

The developer can be empowered to perform static analysis of their code when it is ready and fix the code level issues identified by the scanner. By doing so, dependency of security team is minimized, and bugs that might turn into security vulnerability are fixed before becoming unmanageable.

In the case of iOS application, **Clang**, a static analyzer, is in-built with Integrated Development Environment (IDE) called Xcode. In the case of Android, user would have to install a plugin for integrating static analysis tool with Eclipse IDE. **Findbugs** is one of the free open-source tools available for the Android environment.

Developer training

An IT organization striving to deliver secure applications (including mobile) must engage their developers and train them in secure coding practices. The focus must include delivering a security risk-free application apart from the functionalities and features.

Below are some of the free resources to impart the awareness on secure coding, and developer training to avoid or minimize common security programming errors found in the application.

- Secure coding practices with **Game of hack**. This game educates developers on OWASP Top 10 for 2013 (focuses on web application security, it would be a good resource though for mobile developers also) - <https://www.checkmarx.com/2015/01/20/secure-coding-with-game-of-hacks/>
- Damn Vulnerable iOS App (DVIA) – as the name suggests is a vulnerable mobile application. Main objective of the app is to teach the developers and security enthusiasts about vulnerabilities in mobile apps based on OWASP Top 10 mobile risks - <http://damnvulnerableiosapp.com/>
- Top 25 Most Dangerous Software Coding errors and remediation recommendations from SANS/MITRE - <http://www.sans.org/top25-software-errors/>
- The Software Assurance Forum for Excellence in Code (SAFECode) publication on secure software development and short training videos - <http://www.safecode.org/>
- When an organization is encouraging the security/development professionals to take industry certification on implementing secure coding practices, ISC² certification 'Certified Secure Software Lifecycle Professional (CSSLP)' would be a recommended one.

Testing

In order to continuously integrate security in development, it is important to incorporate security testing along with Quality Assurance tests. One point to remember here is that we are trying to validate the secure development of an application with security testing, as opposed to penetration testing. The latter is a method to intentionally break the security of the application to demonstrate attack vectors available to hackers, and to urge the organizations to fix them.

Usually the Quality Assurance and testing team has a list of test cases to assess the application from a wide range of functional parameters like performance, system, integration, etc. In similar lines, it is essential to create test cases for security testing as well.

Now a natural question arises as to who would perform security testing? It may be performed by the security team or by the development team on the supervision and guidance of the former.

Decentralizing security ownership can truly take place when teams associated with development are empowered and trained to perform the security testing. Also, when these tests are vested with development teams, it reduces back and forth communication to resolve the issues.

However, development teams are not security experts. Results from the security testing performed by development organizations must be vetted by the security organization prior to release of the applications for general use.

We aim to perform Dynamic security analysis in this phase, which is performed against a running instance of a program. This test would most accurately mimic how a malicious user would attack the application. Discovering vulnerabilities can take longer. We need a testing environment to effectively carry out the assessment.

Setting up a Mobile Testing lab

1. Network connection is an essential part of the testing lab. This environment must be isolated from the corporate or production network. Creating a Wi-Fi hot-spot using 3G/4G data card is an option. It is important to remember that both the analysis laptop and the device in which the mobile application is installed need to connect to the same network for some of the tests.

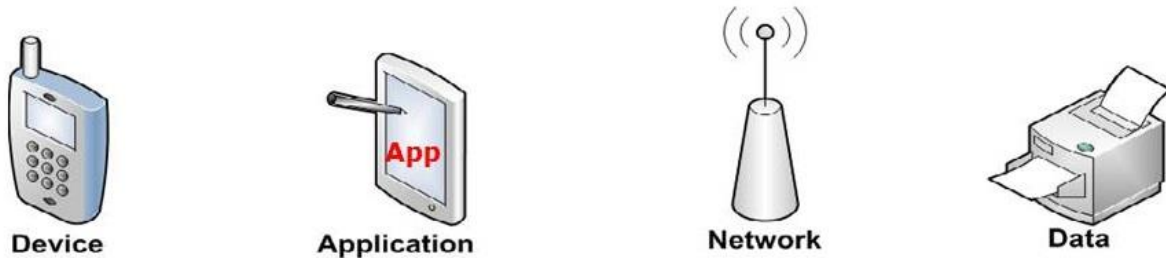
2. Mac or Windows laptop loaded with open-source security software (which are listed below and mapped with security test cases)
3. Jailbroken device for iOS application security testing (iPhone, iPod, or iPad). Refer to - <http://highaltitudehacks.com/2013/06/16/ios-application-security-part-1-setting-up-a-mobile-pentesting-platform/> for instructions on setting up an iOS security lab
4. For Android, we would need to download the Android Software Development Kit (SDK) and Eclipse IDE to set up a lab from - <http://developer.android.com/sdk/index.html>, which includes emulators for mimicking the Android device.
5. Below are some helpful URL links for setting up an Android testing lab.
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-1-setting-your-windows-testing-environment>
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-2-setting-your-linux-testing-environment>
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-3-starting-your-emulator-and-configuring-your-pr>
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-4-installing-applications-your-emulator>
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-5-reverse-engineering-apk-file>
 - <http://www.symantec.com/connect/blogs/android-application-security-assessments-part-6-other-tools-and-random-musings>

Establishing Minimum baseline of Security Test cases

Four major components of the mobility need to be covered in the dynamic analysis. These components are (1) device where the mobile application is installed, (2) application itself, (3) network communication between the application and enterprise server, and (4) data handled in the application

It is important to note that, Static & Dynamic Analysis supplements and compliments each other. Performing both would greatly enhance the security posture of an application.

Dynamic Analysis coverage



When establishing mobile application security testing capability, it may not be possible to focus on everything. The best approach is to, “start small and iterate continuously to mature the capability, along the way incorporating lesson learnt into the process”

HP Mobile application security study released in 2013 (highlighted elsewhere in the beginning of the article) can be good starting point when building the test cases for mobile security testing. HP study is emanating from their experience as Software-as-a-Service (SaaS) mobile application security vendor, and experience in assessing more than 2000 applications from various categories ranging from productivity to social networks, representing all regions of the world.

Another source which can contribute in establishing the baseline security test cases is previous security assessment results of applications developed or used in the organization. Last but not least, and most importantly OWASP Top 10 Mobile security risks are another valuable source to refer to when building the test cases:

https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Mobile_Security_Testing

Security objectives to consider when establishing test cases

Refer to the table below for a sample of 10 security objectives to focus on when creating security test cases for our discussion. The major factor contributing to this list is the assumption that Mobile Device Management (MDM) solutions such as VMware's Airwatch is deployed in the environment and Mobile apps are distributed to users whose device is managed and controlled by MDM.

It is important to remember that this is only a baseline. Readers can have different security objectives which are applicable in their environment. My idea is to suggest how to begin the journey of establishing mobile application security capability with open-source and freely available security tools to bolster the security posture of a mobile application.

Ask every security professional, and their response would be unanimous about the budget and funding for their security projects. These security objectives are the best way to demonstrate value in security investment and gain the trust of those in the executive suite and receive their support for resources.

Security Objective	Coverage	Purpose
Insecure Data Storage	Device	To find storage of credentials in PList files or SQLite Database.
Run-time manipulation	App	To find if the application is susceptible to modification of input to be interpreted as a code instruction
File System Analysis	App	To analyze if any sensitive application data is stored insecurely in the device.
Analyzing Network Traffic	Network	To find whether the application trusts any SSL certificate presented while connecting with Enterprise IT infrastructure, resulting in Man-in-the-Middle (MitM) type of attacks
Insecure or broken cryptography	App	To find out whether weak or flawed encryption algorithm is used to secure the

		information
Information Disclosure	App	To find information leakage via logging, sending analytics data to external providers
Improper session handling	App	To find out whether session time-out is set in the application
Binary protection	Data	To find whether the mobile application binary is secure from reverse engineering risks
Privacy violations	Data	To find out whether the application is using excessive permission than necessary
Authentication	Network, App	To find whether authentication is performed in server-side rather than on the client-side (device)

This article restricts the discussion to iOS and Android application as these are two major mobile application platforms though Blackberry and Windows Mobile OS are not far behind in user adoption.

iOS Security Test Cases

ID	Name	Security tools to use
IOS-01	Storage of credentials in Plist File	PuTTY/ WinSCP/ iExplorer, Plist Editor
IOS-02	Storage of credentials in SQLite File	
IOS-03	Failure to use keychain to store credentials	

IOS-04	Storage of sensitive application data on file system	
IOS-05	Client Trusting any SSL certificate presented – expired or invalid	BurpSuite Proxy, or Fiddler
IOS-06	App allows trivial Man-in-the-Middle (MitM) attack	
IOS-07	Connect to HTTPS once, and fall back.	
IOS-08	Application logging sensitive app data	iPhone Configuration utility
IOS-09	App storing their image in public folder rather than app sandbox (App back grounding)	PuTTY/ WinSCP/ iExplorer
IOS-10	Analytics data sent to 3rd Parties	BurpSuite Proxy, or Fiddler
IOS-11	Authentication requests are performed on server-side	
IOS-12	Persistent authentication if implemented not stores user's password on the device.	WinSCP, Python, BinaryCookieReader.py
IOS-13	Hard-code of Crypto-keys in any construct (plaintext, property files, compiled binaries)	IDA Pro, Clutch, Class-dump-z
IOS-14	Use of Insecure and/or Deprecated Algorithms	

IOS-15	Use of Custom Encryption Protocols	
IOS-16	Invalidate sessions on the back-end	Burp Suite Proxy, Introspy
IOS-17	Reset cookies during authentication state changes	
IOS-18	Adequate timeout protection on the backend components	
IOS-19	Code Obfuscation	IDA Pro, Clutch, Class-dump-z
IOS-20	Remove debugging statements & development information	iRET,
IOS-21	Implementation of ASLR PIE, Automatic Reference Counting,	iRET
IOS-22	Privacy violations – access to location, contacts, address book, photo	Snoop-it, iRET
IOS-23	Access to private data	Snoop-it, iRET
IOS-24	Run-time analysis	GDB, IDA pro, Hopper, ClutchMod

Additional security tools and detailed methodologies can be found in OWASP mobile security project - https://www.owasp.org/index.php/IOS_Application_Security_Testing_Cheat_Sheet

Security blog dedicated to mobile security application from a security researcher, Prateek Gianchandani, is a good resource. Prateek's blog describes how to assess an iOS application with detailed steps - <http://highaltitudehacks.com/index.html>,

Security Researcher Pandurangan's blog has video tutorials on performing iOS application security testing - <http://hackingdemystified.com/>. This site might require a registration; I would

recommend doing so to learn security testing. Bonus is all the resources from both these security enthusiasts are free.

Android Security Test Cases

ID	Name	Security tools to use
AN-01	Storage of credentials in device	AXMLPRNT, SQLite Spy, Cookies Manager
AN-02	Storage of credentials in SQLite File	
AN-03	Failure to use keystore to store credentials	
AN-04	Storage of sensitive application data on file system	
AN-05	Client Trusting any SSL certificate presented – expired or invalid	BurpSuite Proxy, or Fiddler
AN-06	App allows trivial Man-in-the-Middle (MitM) attack	
AN-07	Connect to HTTPS once, and fall back.	
AN-08	Application logging sensitive app data	Burp Suite proxy

AN-09	Analytics data sent to 3rd Parties	BurpSuite Proxy, or Fiddler, Secure Code review
AN-10	Authentication requests are performed on server-side	
AN-11	Persistent authentication if implemented not stores user's password on the device.	
AN-12	Hard-code of Crypto-keys in any construct (plaintext, property files, compiled binaries)	Dex2Jar, JD-GUI, FindBugs, Androwarn,
AN-13	Use of Insecure and/or Deprecated Algorithms	
AN-14	Use of Custom Encryption Protocols	
AN-15	Invalidate sessions on the back-end	Burp Suite Proxy, Introsy
AN-16	Reset cookies during authentication state changes	
AN-17	Adequate timeout protection on the backend components	
AN-18	Code Obfuscation	Dex2Jar, JD-GUI, Proguard tool enables obfuscation which is available as part of Android SDK
AN-19	Remove debugging statements & development information	Dex2Jar, JD-GUI,
AN-20	Privacy violations – access to location, contacts, address book, photo	DroidBox, Mercury Drozer Framework,

AN-21	Access to private data	
AN-21	Run-time Analysis	Mercury Drozer Framework,

Additional security testing suites for Android platform are below. These are freely available mobile application security testing platform.

1. VIALAB community Edition from NowSecure - <https://www.nowsecure.com/blog/2014/09/10/introducing-vialab-community-edition/>
2. MobiSec - <http://pen-testing.sans.org/resources/mobisec>
3. Santoku Linux - <https://santoku-linux.com/>
4. Appuse - <https://appsec-labs.com/appuse/>

Remediation of security issues

Finding vulnerabilities and security flaws is the first step in developing secure software. This process is incomplete unless all the identified security issues are properly tracked and remediated. Issues identified by static analysis, dynamic analysis, and security team review findings needs to be managed in a single repository for various management purposes like metrics, looking for trends, and consolidated risk view of the applications state in an organization.

Since our objective is to continuously integrate security in development of mobile applications with open-source tools, we can take a look at a tool called **Threadfix community edition** from Denim Group for vulnerability tracking and remediation. <http://www.threadfix.org/download/>

Maintenance

Security is an ongoing task; it continues to be important even when the application is released for public use. Proactively monitoring the security vulnerabilities in platform system software and embedded components, and then initiating incidence response and remediation as appropriate is crucial as well.

Identifying security vulnerabilities using reputable sources for obtaining security information is a continuous cycle. Sources such as software vendor websites, NIST's NVD, and MITRE's CVE are reliable for vulnerability research.

Inventory of all third party frameworks/APIs used in the mobile application (discussed earlier in coding section) will be helpful to handle security patches. Whenever any vulnerability becomes public knowledge, a corresponding security update must be done for the mobile applications which are using these vulnerable third party APIs/frameworks.

Appendix I: Steps to execute some of the iOS test cases

Test Case ID: IOS-01

Objective : To find storage of credentials in PList files or SQLite Database. This can also incidentally detect insecure storage of sensitive information within the device

Pre-requisites

1. Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop

Tools needed

PuTTY/ WinSCP/ iExplorer, Plist Editor

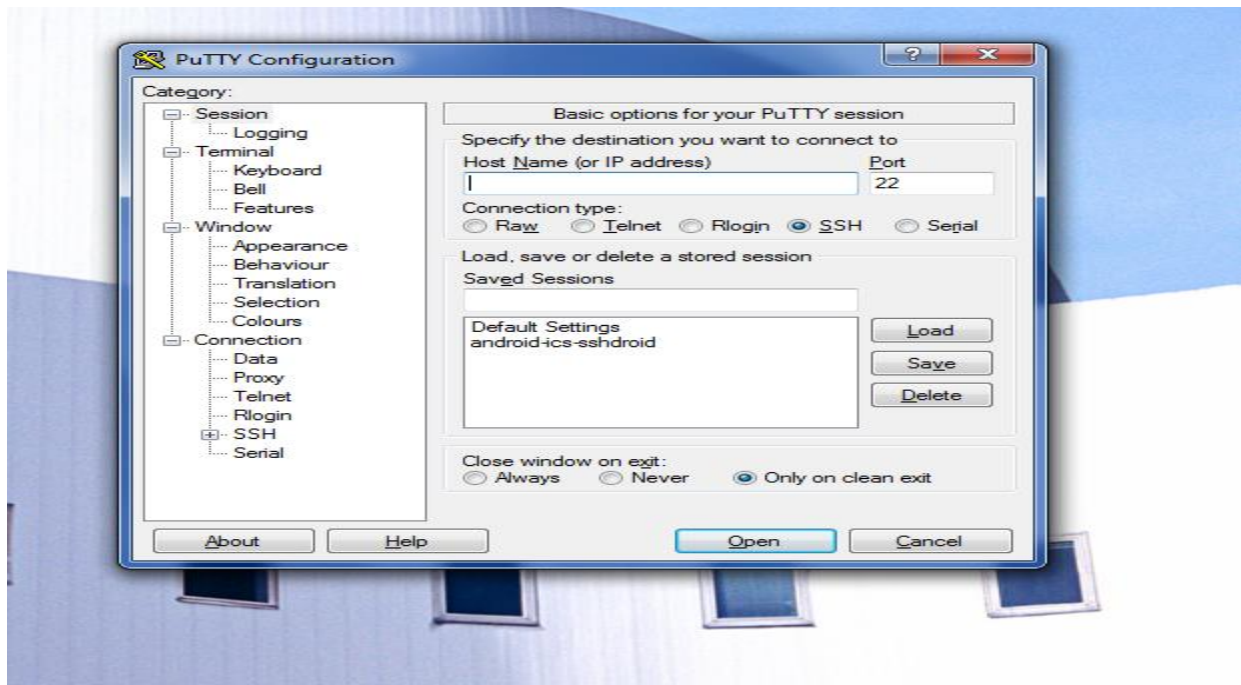
Steps to follow

Testing objective can be achieved by copying the target files from the iDevice and performing the analysis locally in Mac or Laptop. Files can be copied in three ways – using PuTTY, WinSCP, or iExplorer tool, each described below.

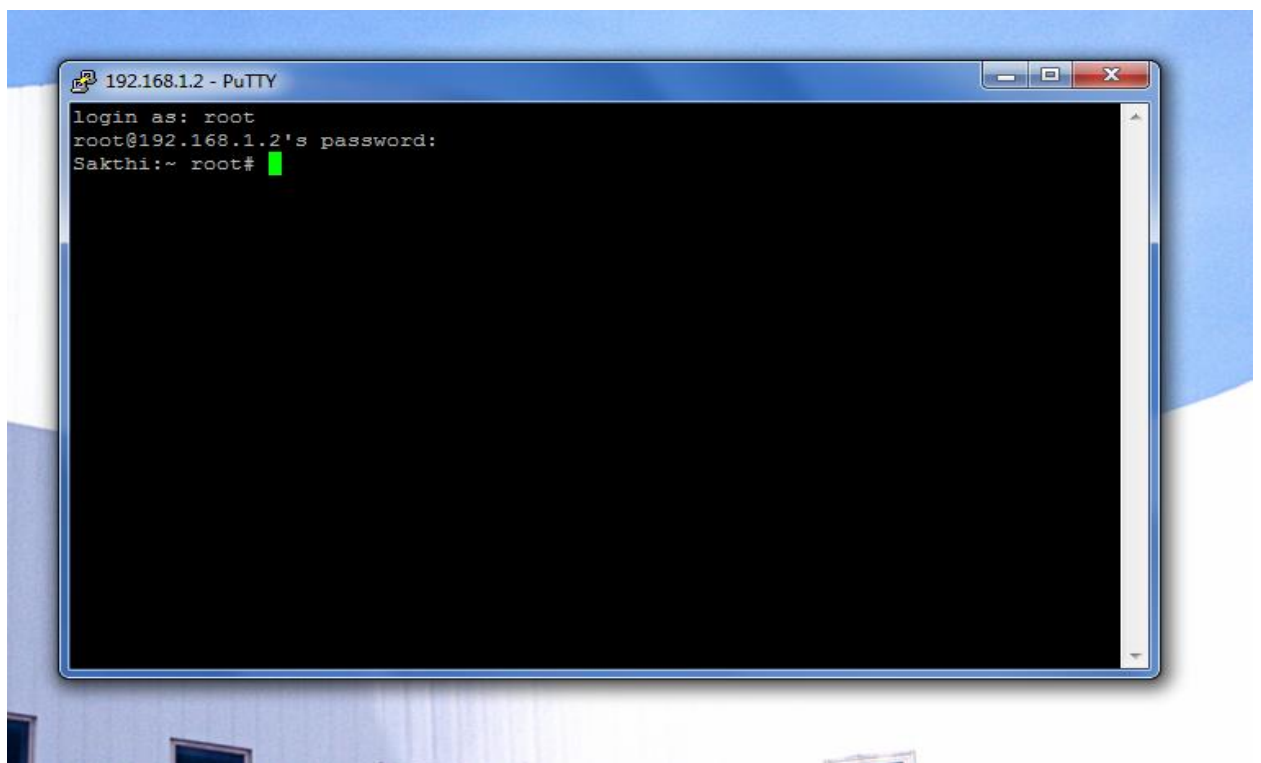
Once the files are successfully brought into the Mac or Laptop, we can perform the security analysis.

Using PuTTY

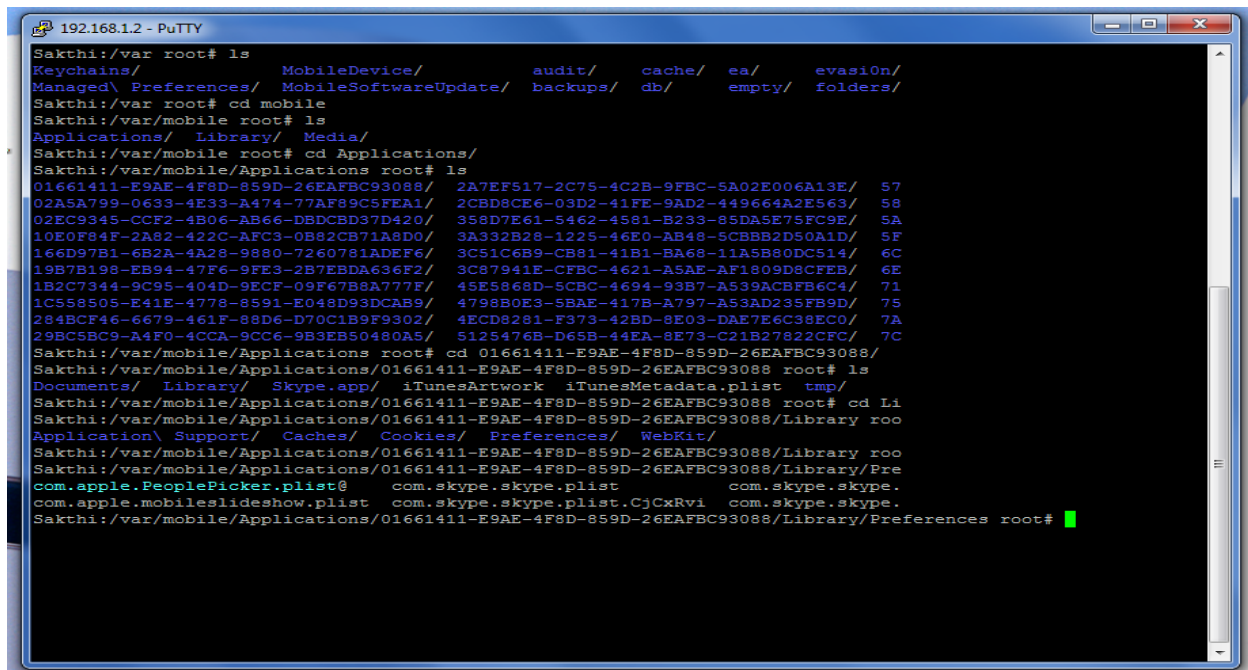
1. Start the PuTTY client and input the IP address of the iDevice in the space provided under 'Host Name (or IP address)', example: 192.168.1.1. Select 'SSH' in connection type. Click 'Open' when all the information is provided as directed



2. PuTTY client will prompt for credentials to log in the iDevice. Log in as 'root' in the Jailbroken iDevice.



3. Navigate to 'root/private/var/mobile/Applications' directory. All the apps downloaded from App store are installed in this location.



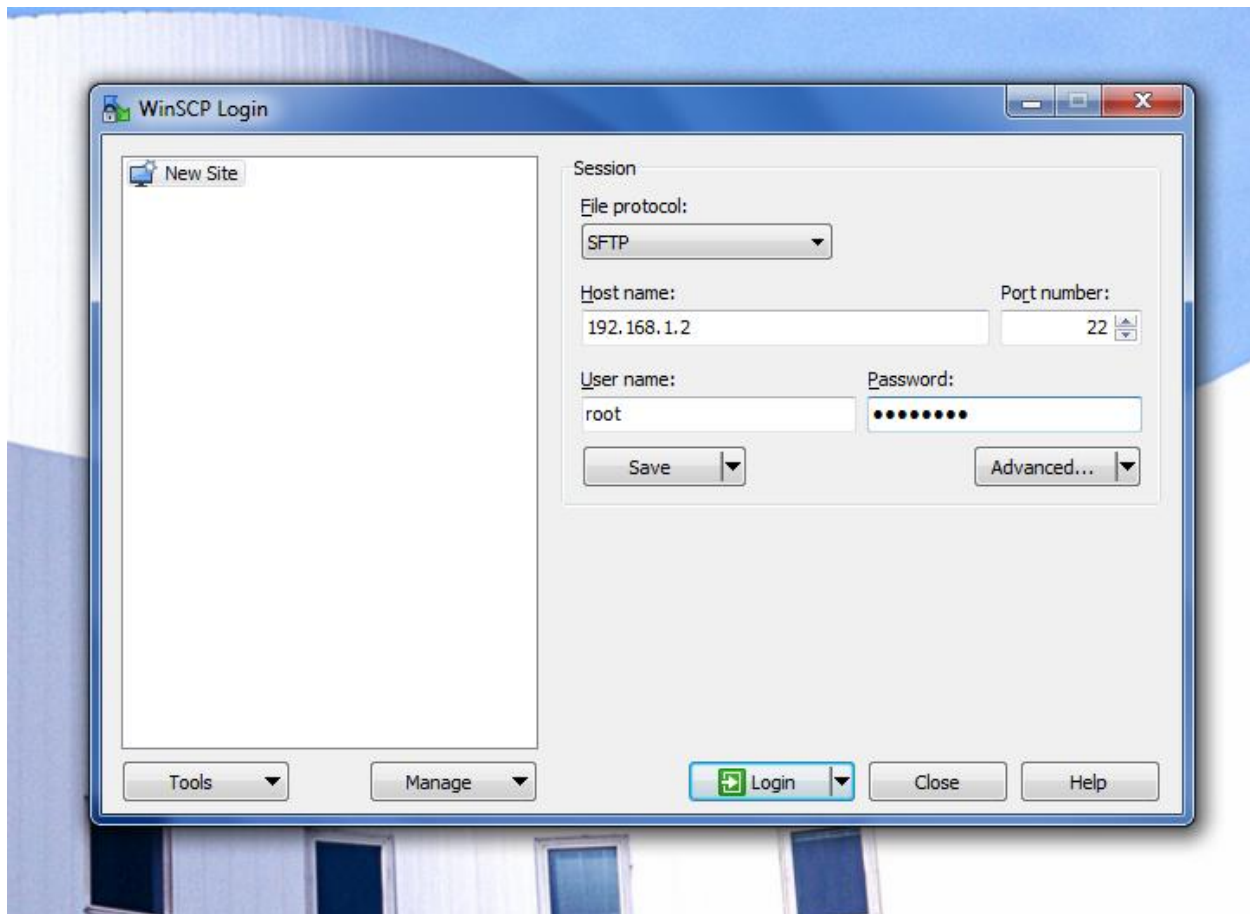
```
192.168.1.2 - PuTTY
Sakthi:/var root# ls
Keychains/      MobileDevice/  audit/         cache/         ea/            evasiOn/
Managed\ Preferences/ MobileSoftwareUpdate/ backups/       db/            empty/         folders/
Sakthi:/var root# cd mobile
Sakthi:/var/mobile root# ls
Applications/  Library/  Media/
Sakthi:/var/mobile root# cd Applications/
Sakthi:/var/mobile/Applications root# ls
01661411-E9AE-4F8D-859D-26EAFBC93088/  2A7EF517-2C75-4C2B-9FBC-5A02E006A13E/  57
02A5A799-0633-4E33-A474-77AF89C5FEA1/  2CBD8CE6-03D2-41FE-9AD2-449664A2E563/  58
02EC9345-CCF2-4B06-AB66-DBDCBD37D420/  358D7E61-5462-4581-B233-85DA5E75FC9E/  5A
10E0F84F-2A82-422C-AFC3-0B82CB71A8D0/  3A332B28-1225-46E0-AB48-5CBBB2D50A1D/  5F
166D97B1-6B2A-4A28-9880-7260781ADEF6/  3C51C6B9-CB81-41B1-BA68-11A5B80DC514/  6C
19B7B198-EB94-47F6-9FE3-2B7EBDA636F2/  3C87941E-CFBC-4621-A5AE-AF1809D8CFEB/  6E
1B2C7344-9C95-404D-9ECF-09F67B8A777F/  45E5868D-5CBC-4694-93B7-A539ACBFB6C4/  71
1C558505-E41E-4778-8591-E048D93DCAB9/  4798B0E3-5BAE-417B-A797-A53AD235FB9D/  75
284BCF46-6679-461F-88D6-D70C1B9F9302/  4ECD8281-F373-42BD-8E03-DAE7E6C38EC0/  7A
29BC5BC9-A4F0-4CCA-9CC6-9B3EB50480A5/  5125476B-D65B-44EA-8E73-C21B27822CFC/  7C
Sakthi:/var/mobile/Applications root# cd 01661411-E9AE-4F8D-859D-26EAFBC93088/
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088 root# ls
Documents/  Library/  Skype.app/  iTunesArtwork  iTunesMetadata.plist  tmp/
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088 root# cd Library
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088/Library root# ls
Application\ Support/  Caches/  Cookies/  Preferences/  WebKit/
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088/Library root# cd Preferences
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088/Library/Preferences root# ls
com.apple.PeoplePicker.plist  com.skype.skype.plist  com.skype.skype.plist
com.apple.mobileslideshow.plist  com.skype.skype.plist.CjCxRvi  com.skype.skype.plist
Sakthi:/var/mobile/Applications/01661411-E9AE-4F8D-859D-26EAFBC93088/Library/Preferences root#
```

4. This directory consists of sub-directories with unique names like "5AA3DFC3-22FA-4E7F-932F-D1BE4B221559". Each application installed in the device has a folder. It is little manual work to find the right folder related to the application being assessed.
5. Navigate further to the 'Library/Preferences' directory to find all the Plist Files related to the application being assessed. Copy the files from here to your Mac or Laptop.

Using WinSCP (FTP Client)

In my case I used the WinSCP FTP client. However, other FTP clients will work in the same way as described here.

1. Start the FTP client and give the IP address of the device (Settings → WiFi → Your network connection), followed by credentials to log in to the device.



2. Navigate to *“/root/private/var/applications/unique ID folder of the application/Library/Preferences”*

Applications		
<div> <div> <div>Applications</div> <div> <div>/ <root></div> <div>private</div> <div>var</div> <div>mobile</div> <div>Applications</div> </div> </div> <div> <div>5AA3DFC3-22FA-4E7F-932F-D1BE4B221559</div> <div>1B2C7344-9C95-404D-9ECF-09F67B8A777F</div> <div>29BC5BC9-A4F0-4CCA-9CC6-9B3EB50480A5</div> <div>C4C9818A-CF4E-4808-BD7E-87DF8D4523C7</div> <div>19B7B198-EB94-47F6-9FE3-2B7EBDA636F2</div> <div>DE3D0298-E2D2-433D-8E16-1640EB8F944A</div> <div>577F4AA9-1B54-45F7-9A4D-6DCDB01C3EF7</div> <div>3C51C6B9-CB81-41B1-BA68-11A5B80DC514</div> <div>AC21C794-5134-4BF0-9504-0F87ED364296</div> <div>753FFE13-2879-4D69-BF71-BFA417F92102</div> <div>10E0F84F-2A82-422C-AFC3-0B82CB71A8D0</div> <div>166D97B1-6B2A-4A28-9880-7260781ADEF6</div> <div>A309D5A3-59AB-47BA-849C-E402C1B96EE2</div> <div>A9DFEF94-455B-49C3-B41C-DF13248CE6B</div> <div>7C3D90AE-BB51-4D63-8725-A886DFB4FB7A</div> <div>BD5BE557-630D-4D1B-AB84-9A99D9F77CEB</div> <div>1C558505-E41E-4778-8591-E048D93DCAB9</div> <div>6C984116-4B6D-42E2-9979-FC9BEC254583</div> <div>358D7E61-5462-4581-B233-85DA5E75FC9E</div> <div>5F574C38-D626-4D4B-A35A-77287CE5D9F0</div> <div>3A332B28-1225-46E0-AB48-5CB8B2D50A1D</div> <div>87DA2D3C-D74F-4F7A-A214-FD30377D391D</div> <div>2CBD8CE6-03D2-41FE-9AD2-449664A2E563</div> <div>2A7EF517-2C75-4C2B-9FBC-5A02E006A13E</div> <div>99887F1F-30E5-4BFD-8D1F-BF7A403D8417</div> <div>02A5A799-0633-4E33-A474-77AF89C5FEA1</div> </div> </div>	Size	Changed
		8/17/2014 10:35:52 PM
		9/18/2014 5:21:20 PM
		9/4/2014 11:09:18 PM
		8/25/2014 11:09:57 PM
		8/24/2014 1:53:24 AM
		8/21/2014 1:13:07 AM
		8/17/2014 10:18:30 PM
		8/17/2014 4:44:23 AM
		8/17/2014 4:42:15 AM
		8/17/2014 4:33:00 AM
		8/17/2014 4:27:47 AM
		8/17/2014 4:25:31 AM
		8/17/2014 4:20:50 AM
		8/17/2014 3:40:18 AM
		8/17/2014 1:14:53 AM
		8/17/2014 1:14:48 AM
		8/14/2014 4:47:25 PM
		8/14/2014 4:29:21 PM
		9/11/2013 10:33:15 PM
		9/11/2013 10:30:57 PM
		9/11/2013 10:28:56 PM
		9/11/2013 10:27:10 PM
		8/31/2013 7:56:44 AM
		8/28/2013 8:27:36 PM
		8/24/2013 8:22:26 AM
		8/24/2013 8:16:54 AM
		8/24/2013 8:12:31 AM

Preferences





Download

Edit

Properties

Find Files

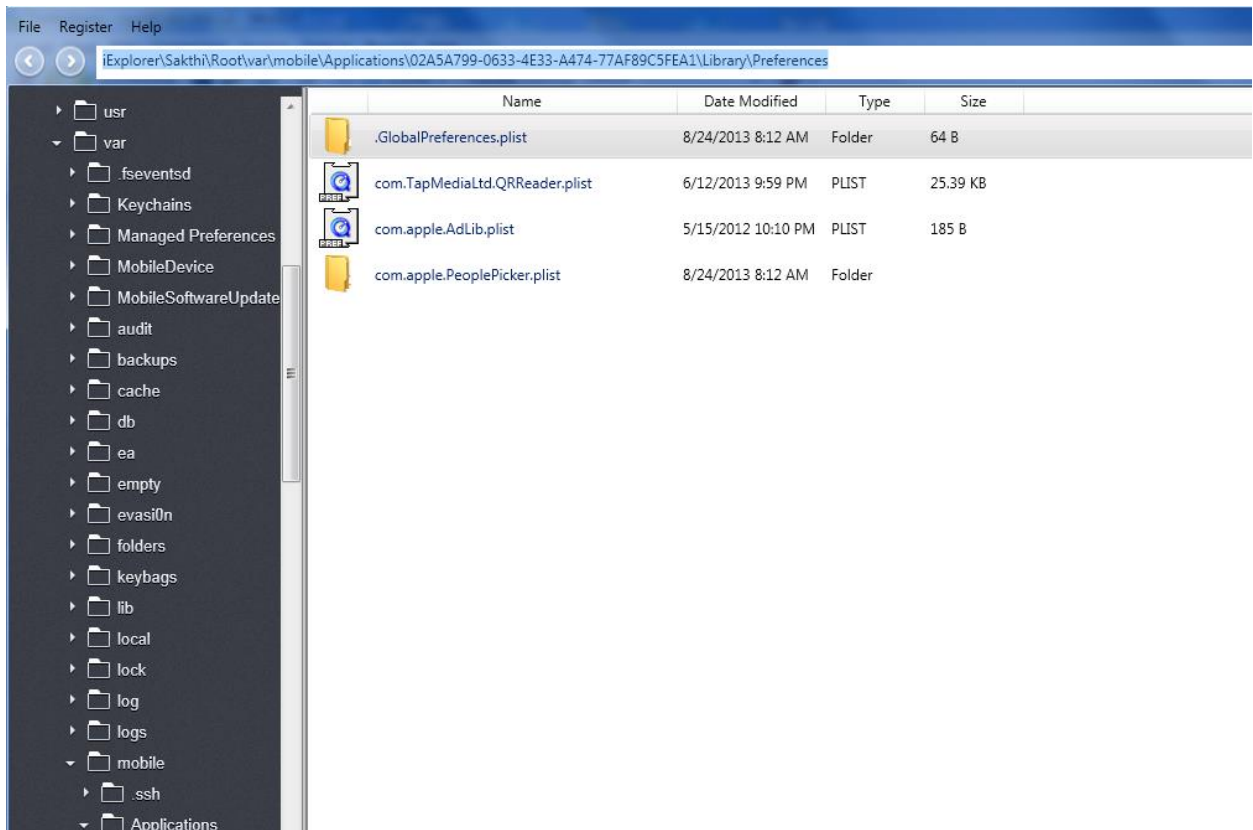
/private/var/mobile/Applications/5AA3DFC3-22FA-4E7F-932F-D1BE4B221559/Library/Preferences

Name	Ext	Size	Changed
			9/18/2014 7:20:14 PM
	com.microsoft.lync2010.iphone.plist	217 B	10/6/2014 11:21:21 PM
	com.apple.PeoplePicker.plist	68 B	9/18/2014 5:21:20 PM
	.GlobalPreferences.plist	64 B	9/18/2014 5:21:20 PM

- Copy the Plist files to Mac or Laptop for further analysis. This can be done by dragging the Plist file from right pane and dropping it on the local folder appearing on the left.

Using iExplorer

1. Using iExplorer is very simple. Connect the iDevice to the USB of Mac or Laptop. Navigate to “/root/private/var/applications/unique ID folder of the application/Library/Preferences”
2. Copy the Plist Files manually to the Mac or Laptop manually.



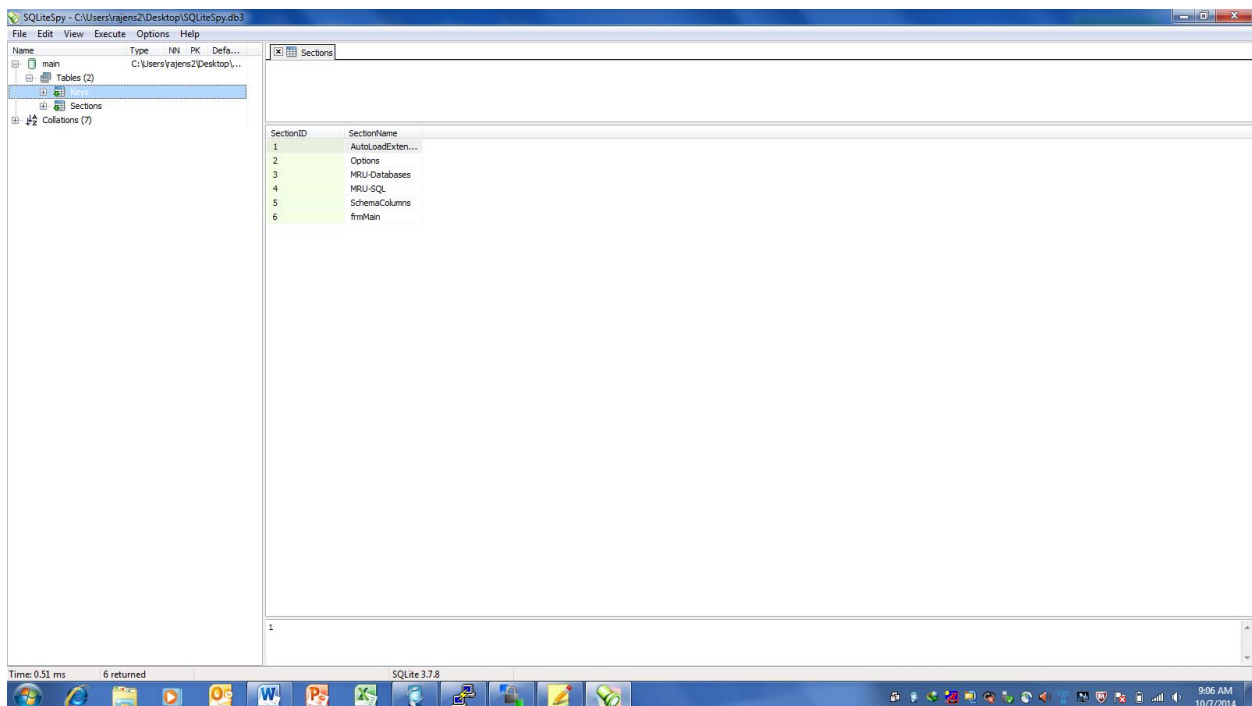
Using Plist Editor to analyze the contents

1. Open the copied *plist file* from your Mac or Laptop using “*Plist Editor*” tool and review the contents for sensitive information, including credentials.

Testing objective can be achieved by copying the target files from the iDevice and performing the analysis locally in Mac or Laptop. Files can be copied in three ways – using PuTTY, WinSCP, or iExplorer tool, as described in Test Case ID *IOS-01*

Once the Database files are successfully brought into the Mac or Laptop, we can perform the security analysis.

1. Navigate to “/root/private/var/applications/unique ID folder of the application/Documents” directory and locate the SQLite Database file. Follow the steps to copy the Database file to Mac or Laptop using PuTTY, WinSCP, or iExplorer.
2. Open the Database file using SQLiteSpy and analyze the contents to find any sensitive information.



Test Case ID: IOS-05

Objective: To find whether the application is trusting any SSL certificate presented while connecting to Enterprise IT infrastructure. For example, this test case intends to assess whether the network communication is secure when a user is connecting from a public Wi-Fi hot spot.

Pre-requisites

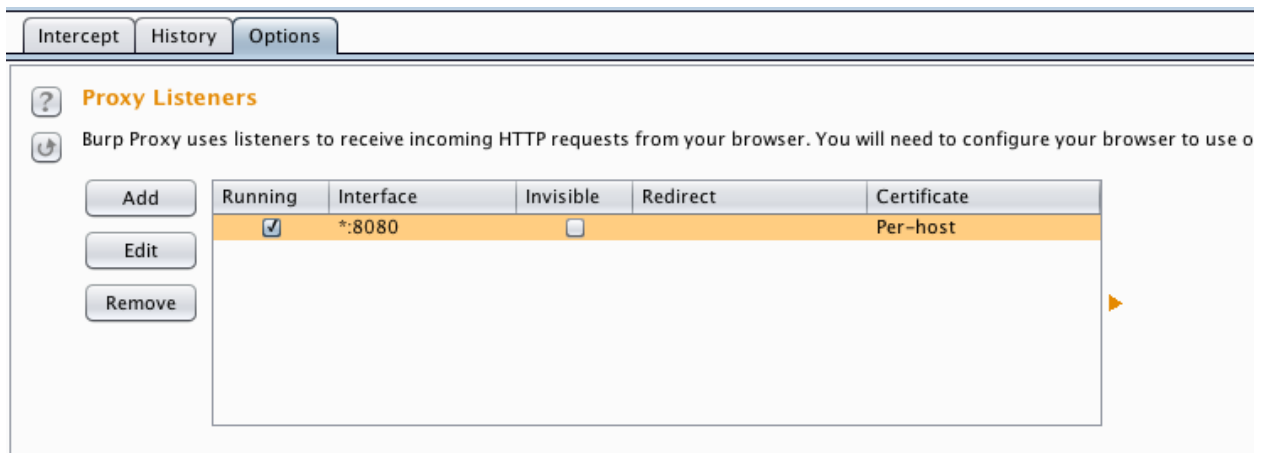
1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Almost similar to Test Case ID *IOS-01*

Tools needed

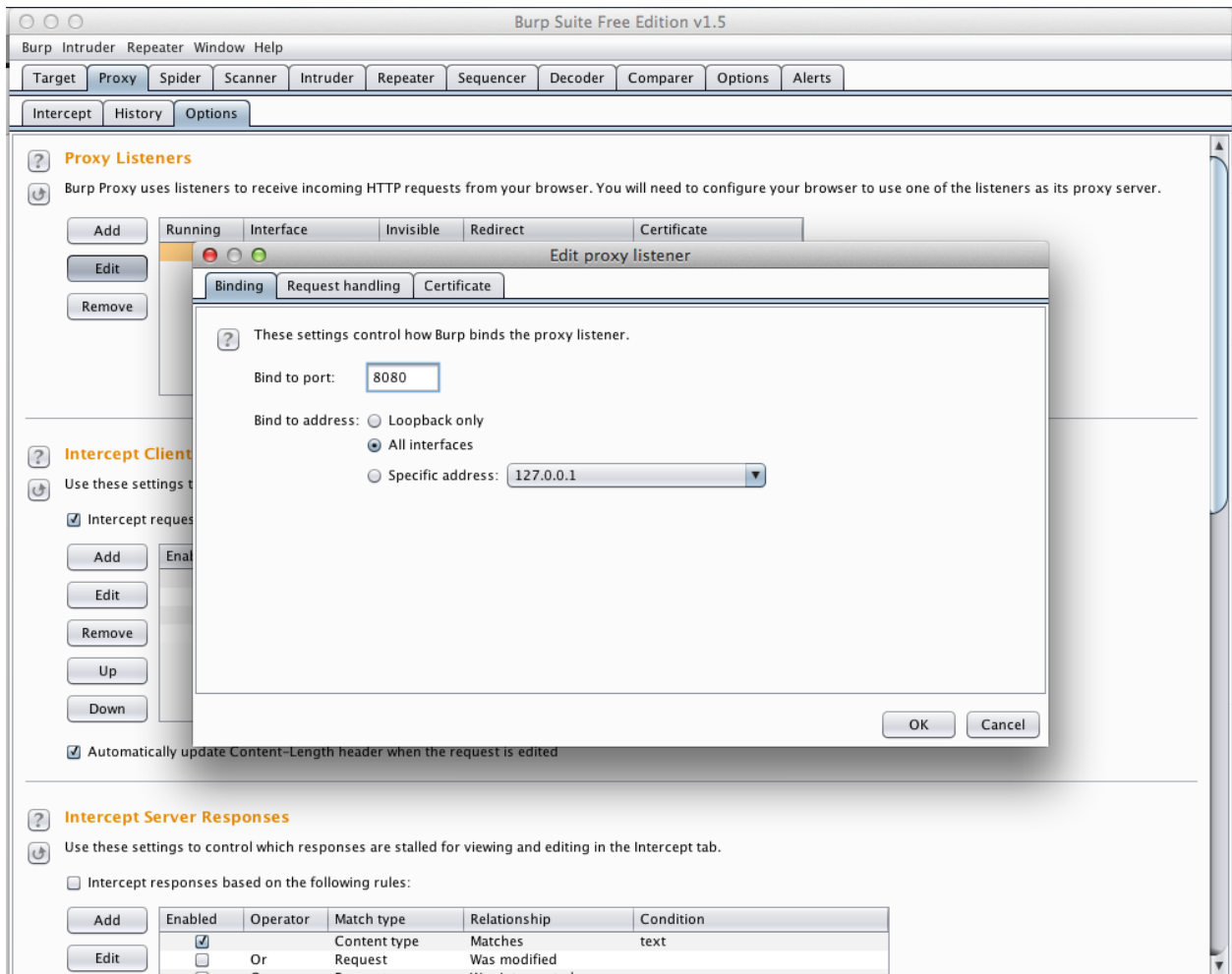
BurpSuite Proxy

Steps to follow

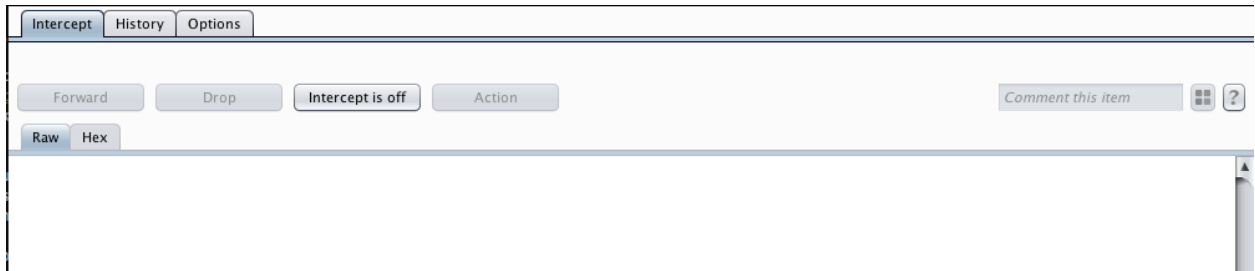
1. Open Burpsuite from your Mac or Windows laptop, go to *Proxy* → *options*



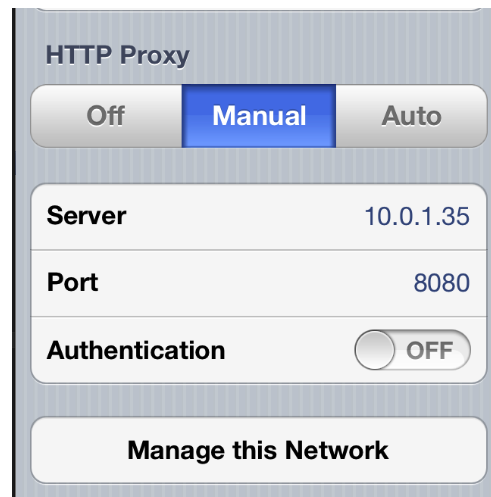
2. Click on the proxy that is set, click on *Edit* and select the option *All Interfaces* inside the option *Bind to Address*



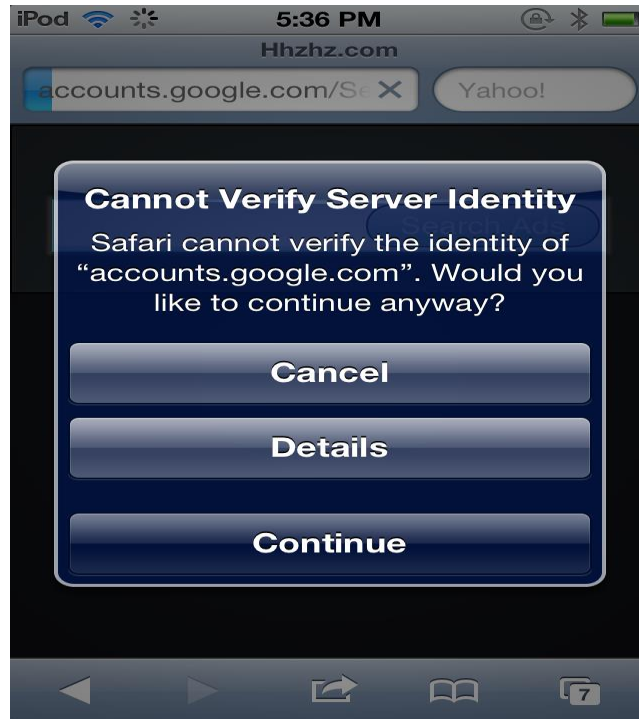
3. By default, Burp creates a self-signed CA certificate upon installation. The current checked option (when you click on *Certificate tab*), i.e. *generate CA-signed per-host certificates* - will generate a certificate for the particular host we are connecting to signed by the CA certificate that was created on installation of Burp. Click 'OK' to move to next step.
4. Now, go to *Proxy -> Intercept* and make sure Intercept is set to Off. This is because you may not want to be bothered with forwarding every packet that comes through the proxy.



- Now, you will have to configure your device to route traffic through this proxy. On your iOS device, go to the Settings App, click on *Wifi*, select the network that you are connected to and on its settings, scroll down and there will be an option to set its proxy. Set the proxy as your computers' IP address which is currently running Burpsuite and the port as the port on which the proxy is running.

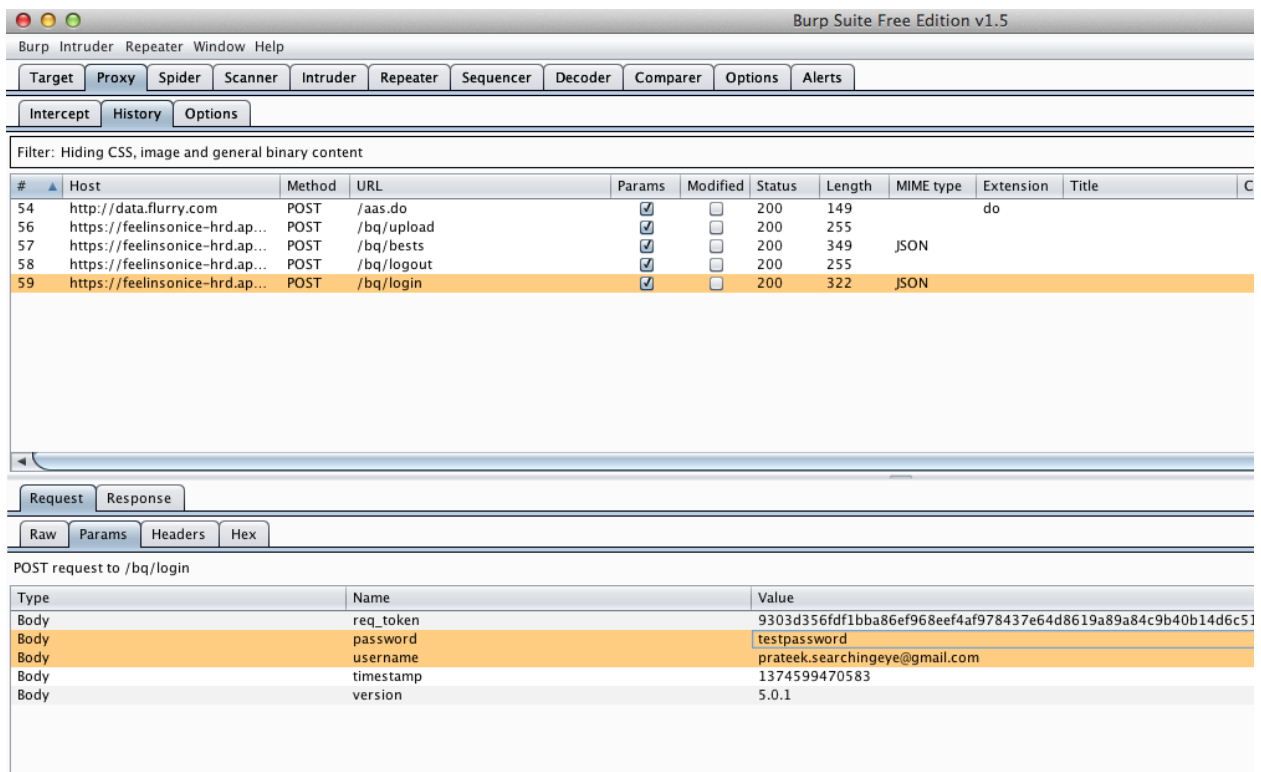


- Now start using an application from your device. Some applications are coded in such a way that only SSL connections are allowed. However, some applications will issue a warning and ask to confirm or cancel the connection. For instance, the screenshot below shows a warning when run using the proxy.



If you click on *Continue*, you will be able to see the application traffic. Please note that this warning is accepted only for this host currently. If you browse to another website with HTTPS enabled, you will be shown another warning because Burpsuite generates a fake SSL certificate for every host.

7. This traffic will be intercepted by Burpsuite. As you can see from the figure below, in the login call, we can see the username and password being sent as well as other API calls made by the application.



Note: However, one issue is that when a new application is accessed from the device, it would prompt the certificate warning again. We can overcome that by making the Burp's certificate as trusted root certificate in the device. We might need to copy the certificate from the laptop or Mac and install in the device to avoid warning prompts.

Test Case ID: IOS-08

Objective: To find whether the application is logging sensitive data in the logs.

Pre-requisites

1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Similar to Test Case ID *IOS-01*

Tools needed

iPhone Configuration utility

Steps to follow

In general, iOS applications write data into logs for diagnostic and troubleshooting purposes. Also, during development, applications developers commonly use NSLog for debugging purposes. These logs might include requests, responses, cookies, authentication tokens, and other sensitive data.

To view the error logs,

1. On Windows, Install iPhone Configuration Utility.
2. Connect the iPhone to the Windows using USB cable.
3. Select the connected device in the iPhone Configuration utility, click on Console table to view the application logs.

Tip:

As we saw earlier in Test cases IOS-01, apps installed on the iPhone are located at */private/var/mobile/Applications/[unique id] directory*. Typical iPhone application home directory structure is listed below.

SubDirectory	Description
Appname.app	Contains the application code and static data
Documents	Data that may be shared with desktop through iTunes
Library	Application support files
Library/Preferences/	App specific preferences
Library/Caches/	Data that should persist across successive launches of the application but not needed to be backed up
tmp	Temporary files that do not need to persist across successive launches of the application

<https://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGuide/FileSystemOverview/FileSystemOverview.html>

Test Case ID: IOS-09

Objective: To find where the application is storing the image when a user is trying to take a screenshot when the application is running. It is known as App background. Our aim is to assess where the app stores the image because storing it in a public folder when the screenshot has sensitive info is not desirable.

Pre-requisites

1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Similar to Test Case ID *IOS-01*

Tools needed

PuTTY/ WinSCP/ iExplorer

Steps to follow

1. Take a screenshot when the application is running
2. Navigate to Library/Caches/Snapshot folder of the application home directory
3. If the image is not found, you might check in public photo gallery to see if the screenshot is present. If you find the image in this location, then it is an issue because the app might store sensitive information in the device without user's knowledge.

Test Case ID: IOS-10

Objective: To find whether the application is sending sensitive data to 3rd party sites for analytics purpose.

Pre-requisites

1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Successfully setting up the Burp suite Proxy as in Test Case ID *IOS-05*

Tools needed

BurpSuite Proxy,

Steps to follow

1. Run the application and monitor the traffic in Burp Suite Proxy to find the data transmission and identify the analytics data being sent.

Test Case ID: IOS-11

Objective: To find whether the application is performing authentication in the server-side as opposed to client-side validation.

Pre-requisites

1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Successfully setting up the Burp suite Proxy as in Test Case ID *IOS-05*

Tools needed

BurpSuite Proxy,

Steps to follow

1. Run the application and monitor the traffic in Burp Suite Proxy to find the data transmission and identify whether the authentication information is sent to the external server, instead of authentication locally within the client device

Test Case ID: IOS-12

Objective: To find whether the authentication information is stored in cookies if persistent authentication is implemented.

Pre-requisites

1. Normal or Jailbroken iDevice with iOS application be tested installed on it
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop
4. Successfully setting up the Win SCP as in Test Case ID *IOS-01*
5. Environment is set up to run Python

Tools needed

WinSCP, Python, BinaryCookieReader.py (*download from <http://securitylearn.net/wp-content/uploads/tools/iOS/BinaryCookieReader.py>*)

Steps to follow

1. Make sure your device and Mac or Laptop is connected to the same network
2. Run WinScp and SSH into the iPhone by typing the iPhone IP address from your Mac or Laptop
3. Navigate to the Library/Cookies folder in the application's home directory.
4. Copy the Cookies.binarycookies file to the Windows machine by dragging it
5. On Windows or Mac, open command prompt and run the below command to list the contents of cookies.binarycookies file.

Python BinaryCookieReader.py [file path of Cookies.binarycookies]

Test Case ID: IOS-19

Objective: This test case aims to automate many of the manual tasks needed to perform in order to analyze and reverse engineering iOS applications.

Pre-requisites

1. Jailbroken iDevice with iOS application be tested installed on it. *Refer to <https://www.veracode.com/blog/2014/03/introducing-the-ios-reverse-engineering-toolkit> for dependencies to be installed from Cydia and general overview of the tool*
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop

Tools needed

iRET, Python

Steps to follow

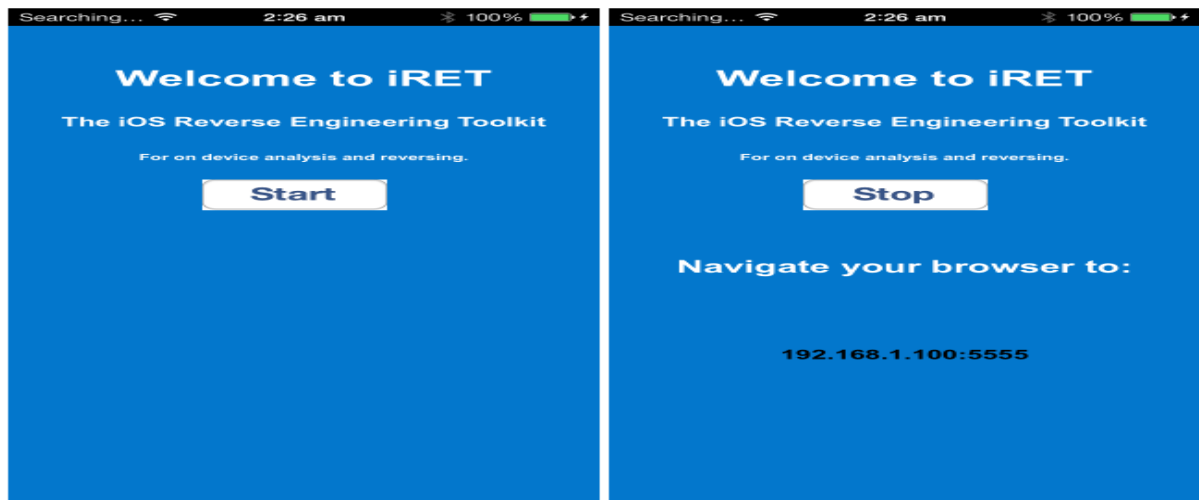
1. Download iRET from <https://www.veracode.com/sites/default/files/Resources/Tools/iRETTool.zip> on the device directly or file transfer to the device using WinSCP from your laptop or Mac
2. Unzip the file as shown below

```
iPhone:~ root# unzip iRETTool.zip
Archive: iRETTool.zip
  inflating: iRET-Tool/iRET.deb
  inflating: iRET-Tool/README
  extracting: iRET-Tool/Source_Code.zip
Prateeks-iPhone:~ root# cd iRET-Tool/
Prateeks-iPhone:~/iRET-Tool root# ls
README Source_Code.zip iRET.deb
iPhone:~/iRET-Tool root#
```

3. Install *iRET.deb* as shown below

```
iPhone:~/iRET-Tool root# dpkg -i iRET.deb
Selecting previously deselected package ire.
(Reading database ... 3831 files and directories currently installed.)
Unpacking ire (from iRET.deb) ...
Setting up ire ("1.0") ...
```

4. Now respring or reboot your device, and you will see a new icon with the name iRET. Tap on it to launch the app. We can start the server and then connect to it via our browser. Tap on Start and navigate our browser to the mentioned address.



- Once accessing the device in your browser, we are provided the interface as shown in the image below. Please note that the first time you navigate the browser to this address, it takes a little bit of time, so be patient.

Welcome to iRET The iOS Reverse Engineering Toolkit

<p style="text-align: center;">What is in the toolkit?</p> <ul style="list-style-type: none"> - oTool (Installed) - dumpDecrypted (Not Installed) - Sqlite (Installed) - Theos (Installed) - Keychain_dumper (Installed) - file (Not Installed) - plutil (Installed) - class-dump-z (Installed) <p style="text-align: center; font-size: small;">Note: All tools listed above must be installed.</p>	<p style="text-align: center;">To begin, select an app from the list below:</p> <div style="border: 1px solid #ccc; padding: 2px; text-align: center;"> Select Application ⌵ </div>
---	---

- On the left hand side, we can see whether the tools needed by iRET to perform its tasks are installed or not. Installing a file is very straightforward. You can download it just by searching via Cydia.
- From the right side, we can select the application that we want to analyze.
- Once the application is selected, iRET gives us assessment info under each of the tabs.

Welcome to iRET

The iOS Reverse Engineering Toolkit

Binary Analysis
Keychain Analysis
Database Analysis
Log Viewer
Plist Viewer
Header Files
Theos
Screenshot
Home

Binary Analysis Results

Below are the results of the otool analysis.

Headers

```

/var/mobile/Applications/33EF9650-E655-4E7D-BB4C-DEE63E00C5B8/DamnVulnerableIOSApp.app/DamnVulnerableIOSApp:
Mach header
magic cputype cpusubtype caps filetype ncmds sizeofcmd flags
MH_MAGIC ARM 9 0x00 EXECUTE 33 3684 NOUNDEFS DYLDLINK TWOLEVEL PIE
          
```

Encryption Info

```

cmd LC_ENCRYPTION_INFO
cmdsize 20
cryptoff 16384
cryptsize 1884160
cryptid 0
          
```

Stack Smashing Info

```

0x001d3ed4 20805 __stack_chk_fail
0x001d4218 20805 __stack_chk_fail
0x001d4348 20806 __stack_chk_guard
          
```

ARC Info

```

0x001d3e34 20901 _objc_release
0x001d4178 20901 _objc_release
          
```

Test Case ID: IOS-22

Objective: This test case allows us to perform runtime analysis and find whether the app can access private data from the user device.

Pre-requisites

1. Jailbroken iDevice with iOS application be tested installed on it. *Refer to <https://code.google.com/p/snoop-it/> to download the tool and for general overview.*
2. Macbook or Laptop connected to the same WiFi as the iDevice
3. Required tools already installed in the Macbook or Windows laptop

Tools needed

Snoop-it

Steps to follow

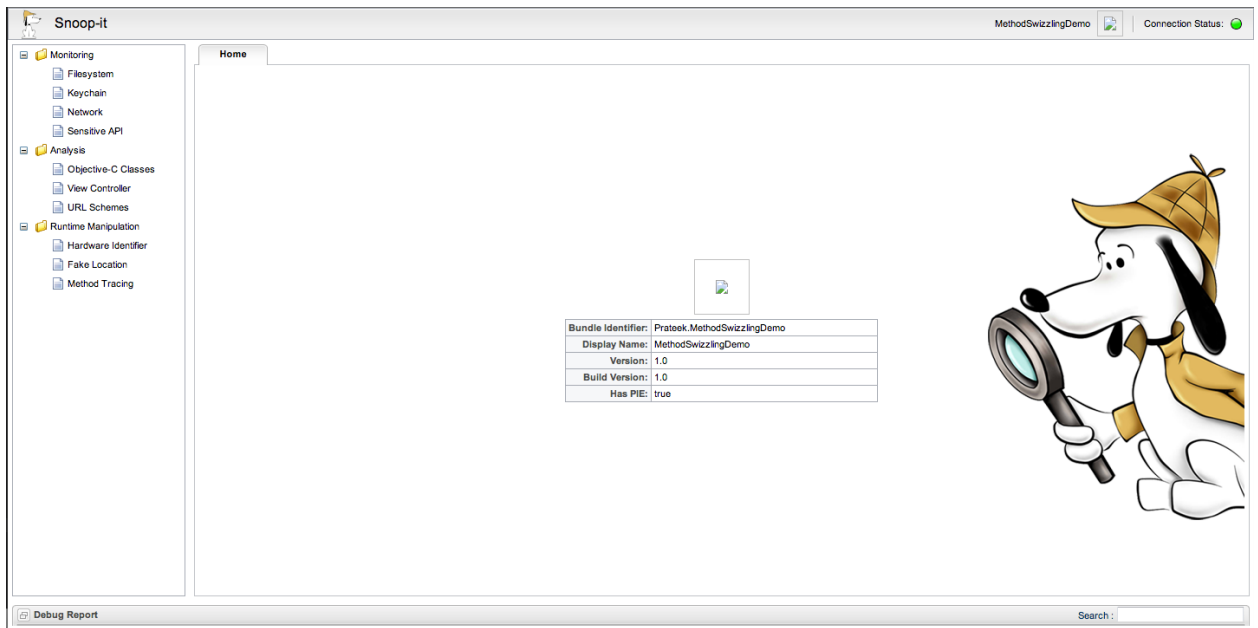
1. Download Snoop-it from its official download page on the device directly or file transfer to the device using WinSCP from your laptop or Mac
2. Install the package using command: `dpkg -i snoop-it_beta8.deb`
3. Once this is done, you will see the Snoop-it app icon on your device. Open it up and you will see the below user interface.



4. Go to Settings and configure the app according to your need. In this case, choose the default port as 12345 and disable the authentication for ease of use.



5. Now, open the Snoop-it web interface in web browser to the address provided on the Snoop-it application. In this case, the address is <http://10.0.1.79:12345>
6. Select an application that needs to be analyzed from the Snoop-it application, open it up on the device, and then refresh the web interface.
7. Now, make sure that the app is opened on your device and in the foreground and refresh the Snoop-it web interface.
8. Perform security assessment of the application from the tabs found in the web interface.



References

1. <http://securityaffairs.co/wordpress/31028/hacking/vulnerabilities-in-alibaba.html>
2. <http://www.latimes.com/business/technology/la-fi-tn-alibaba-security-breach-20141210-story.html>
3. <http://www.zdnet.com/article/kindle-security-vulnerability-can-compromise-amazon-accounts/>
4. <http://gibsonsec.org/snapchat/>
5. <http://www.businessinsider.in/A-Security-Researcher-Finds-A-Weakness-In-The-Starbucks-Mobile-App/articleshow/28960889.cms>
6. <http://seclists.org/fulldisclosure/2014/Jan/64>
7. <http://www.decompilingandroid.com/mobile-app-security/top-10-mobile-security-risks/>
8. <http://www.darkreading.com/vulnerabilities---threats/weak-security-in-most-mobile-banking-apps/d/d-id/1141054?>
9. <http://blog.ioactive.com/2014/01/personal-banking-apps-leak-info-through.html#more>
10. <http://venturebeat.com/2014/11/26/uber-app-collects-an-uncool-amount-of-data-about-users-security-pro-says/>
11. <http://www.gironsec.com/blog/2014/11/what-the-hell-uber-uncool-bro/>
12. <https://www.uber.com/android/permissions>
13. <http://www.telegraph.co.uk/technology/internet-security/11106135/Security-experts-criticise-eBays-response-to-latest-breach.html>
14. <http://www.telegraph.co.uk/technology/internet-security/10849689/eBay-hacking-online-gangs-are-after-you.html>
15. <http://www.tripwire.com/state-of-security/security-data-protection/2014-the-year-of-the-breach-part-1/>
16. <http://www.csoonline.com/article/2842532/data-breach/6-things-we-learned-from-this-years-security-breaches.html>
17. <http://www.zdnet.com/article/hackers-access-800000-orange-customers-data/>
18. <http://www.databreachtoday.asia/south-korean-telecom-co-breached-a-6605>
19. <https://www.trustedsec.com/august-2014/chs-hacked-heartbleed-exclusive-trustedsec/>
20. <http://www.mumsnet.com/info/the-heartbleed-security-breach-to-do>
21. <http://www.bbc.com/news/technology-26969629>
22. <http://www.zdnet.com/article/pf-changs-security-breach-data-stolen-from-33-locations-over-8-months/>
23. <http://www8.hp.com/h20195/V2/GetPDF.aspx/4AA5-1057ENW.pdf>

24. <http://h30499.www3.hp.com/t5/Fortify-Application-Security/HP-Study-Reveals-70-Percent-of-Internet-of-Things-Devices/ba-p/6556284#.VLKPbSuUeYI>
25. <http://h20195.www2.hp.com/V2/GetDocument.aspx?docname=4AA5-4759ENW&cc=us&lc=en>
26. <https://www.isc2.org/GISWSRSA2013>
27. <https://www.isc2cares.org/uploadedFiles/wwwisc2caresorg/Content/GISWS-infographic.pdf>
28. <http://techcrunch.com/2015/01/02/the-cybersecurity-tipping-point/>
29. <http://cybersecurity.ieee.org/center-for-secure-design/avoiding-the-top-10-security-flaws.html>
30. <http://searchsecurity.techtarget.com/opinion/Thirteen-principles-to-ensure-enterprise-system-security>
31. http://www.acsac.org/secshelf/papers/protection_information.pdf
32. https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Mobile_Security_Testing
33. <https://www.enisa.europa.eu/activities/Resilience-and-CIIP/critical-applications/smartphone-security-1/smartphone-secure-development-guidelines>
34. <https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/SecureCodingGuide.pdf>
35. https://www.owasp.org/index.php/IOS_Developer_Cheat_Sheet
36. <https://www.checkmarx.com/2015/01/20/secure-coding-with-game-of-hacks/>
37. <https://www.checkmarx.com/2014/12/31/ethical-hackers-tips/>

EMC believes the information in this publication is accurate as of its publication date. The information is subject to change without notice.

THE INFORMATION IN THIS PUBLICATION IS PROVIDED "AS IS." EMC CORPORATION MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WITH RESPECT TO THE INFORMATION IN THIS PUBLICATION, AND SPECIFICALLY DISCLAIMS IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Use, copying, and distribution of any EMC software described in this publication requires an applicable software license.