Mobile Security Testing

Bhabani Prasad Swain
Rahul Kumar Sinha
Keshava Murthy
Introduction ................................................................................................................................. 3
The Mobile Ecosystem .................................................................................................................... 3
Challenges .................................................................................................................................... 4
Proposed Solution ........................................................................................................................... 4
Understanding & Formulation of Implicit Security Requirements .................................................. 5
Security vs. Usability Trade-off ....................................................................................................... 5
Information Gathering ..................................................................................................................... 6
Application Footprint Analysis for Native and Hybrid Applications .................................................. 6
Adaptation of Existing Tools and Techniques for Penetration Testing ............................................. 7
Interception of Data Through Proxy .................................................................................................. 7
User Agent Configurable Crawlers .................................................................................................. 7
Parsing Application Logs ............................................................................................................... 8
Client Side Security Testing ........................................................................................................... 8
Ensuring Test Completeness ......................................................................................................... 8
Knowledge Base Management (Paranoid Beehive policy) ............................................................... 10
Target Device Selection ................................................................................................................ 11
Conclusion ..................................................................................................................................... 12
About the Authors .......................................................................................................................... 12
References ..................................................................................................................................... 13
Appendix ........................................................................................................................................ 14
Introduction

This document conceptualizes viable and cost-effective Process, Techniques, and Tools, which will improve efficacy of mobile application security testing. This white paper attempts to provide a structured process, some proven techniques, and a unique way for target device selection that will help in improving standards and reducing the costs of application security testing for mobile platforms that remain severely exposed to cyber threats.

The Mobile Ecosystem

Before we delve any further into mobile application security testing, it is vital to define the ecosystem that makes up the mobile platform from a tester’s perspective.

A mobile platform has 3 salient components:
- The Mobile Device (OS, processor, memory, communication channels, sensors, and interfaces)
- The Network (UMTS, EDGE, GPRS, Wi-Fi, etc.)
- The Destination/Server (web services, webpages, podcasts, streaming, RSS feeds)

Noticeably, a mobile platform corresponds to the classical model of web application security in terms of their components. Drawing an analogy, it is possible to say: the mobile device acts as the “Source”, the channel as the “Network”, and the “Server” as the destination.

The basic functionality of mobile components may resemble the classical client-server architecture, but mobile application security testing is radically different with diverse and subjective challenges.

A comparative table for understanding the application classes from a Tester’s perspective:

<table>
<thead>
<tr>
<th>Native Mobile applications</th>
<th>Mobile Web Application</th>
<th>Hybrid Application</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Definition</strong>: Native mobile apps are installed on the devices directly. They can make use of sensors and hardware components of the device such as Bluetooth, NFC, and camera.</td>
<td><strong>Definition</strong>: Mobile web apps are designed specifically for mobile interfaces. Such applications are optimized for unreliable network connections, small resolutions, touch interfaces, weaker processing power, and low memory.</td>
<td><strong>Definition</strong>: Hybrid apps, as the name suggests, are a hybrid between native and mobile web apps. They are installed in the device memory but they communicate with the server side component for data and other services.</td>
</tr>
<tr>
<td><strong>Primary Threats</strong>: - Information disclosure - Broken authentication - Authentication by-pass - Device integrity failure</td>
<td><strong>Primary Threats</strong>: - SQL injection - Session hijacking - Slowloris attack</td>
<td><strong>Primary Threats</strong>: - Eavesdropping - Reverse engineering - Spoofing - Request forgery - Information disclosure</td>
</tr>
<tr>
<td><strong>Scenario</strong>: (Authentication bypass) Camera app opens from lock screen via a combination of gestures. Once a picture is clicked, it shows the picture in preview mode. The picture taken is allowed to be shared, hence bypassing the locked phone’s authentication fence.</td>
<td><strong>Scenario</strong>: (Slowloris attack) As mobile web apps have to deal with unreliable network connections, the average connection timeout is kept a bit longer than necessary. This may create an opportunity for application-layer DoS attacks such as slow HTTP post attacks.</td>
<td><strong>Scenario</strong>: (Eavesdropping) Due to constraints in mobile-device processing power, the encryption standard used might be weak. While using public WIFI connections, the traffic can be analyzed and decrypted easily.</td>
</tr>
</tbody>
</table>
Challenges

Mobile application security testing involves some unique challenges:

- Limited budget and time constraints make mobile application security testing costly.
- Security requirements are seldom complete and well defined.
- Industry standards don't exist for the quality of mobile application security.
- Traditional testing paradigm is insufficient for specific technical challenges in mobile application testing.
- Mobile application security testing labs are expensive to set up and maintain.
- Knowledge management and resource competency are not monitored properly since it is relatively a new technology.

Proposed Solution

Complete mobile security testing requires modifications in the traditional security-test framework. The proposed solution addresses challenges with exclusive focus on concepts significantly different from those in the traditional client-server architecture.

Solutions Outline


   a. Understanding & Formulation of Implicit Security Requirements
      i. Security vs. Usability Trade-Off
   b. Information Gathering Approach for Mobile Platform
      i. Application Footprint Analysis
   c. Adaptation of Existing Tools and Techniques for Penetration Testing of Mobile Platform
      i. Interception of Data Through Proxy
      ii. User Agent Configurable Crawlers
      iii. Parsing Application Logs
      iv. Client Side Security Testing
   d. Ensuring Test Completeness

2. Knowledge Base Management for Mobile Security Projects

3. Target Device Selection


Irrespective of the security testing framework opted, it must employ the basic steps as shown in Figure 2 in some form or another.

![Figure 2: Generic Security Testing Lifecycle](www.tavant.com/testing)
a. Understanding & Formulation of Implicit Security Requirements

Identifying the direct and implied security requirements can change customer-focus strategies. While customer stories are helpful, the onus is on test professionals to identify the implied security requirements.

A synthesized process for the task based on past projects and how it is done:

- Note all positive use cases
- Derive all negative use cases using “use and abuse” methodology
- Document the possible attack surface of the application
- Document security and compliance guidelines for the application
- Conduct a Security vs. Usability Trade-off analysis (before base lining the security requirement)

Implicit Security Requirements:

- The connection to the payment gateway should be encrypted.
- Sensitive information should be ‘secure erased’ or stored with irreversible encryption after the transaction.
- The encryption algorithm used should be of reasonable complexity to provide secure communication without hindering the performance of the application at the same time.
- The transaction should not appear in any form in application logs. The transaction history should always be fetched from server with proper authorization checks, etc.

I. Security vs. Usability Trade-Off

Unlike popular belief, security and usability never find a linear relation of inverse proportionality. If we factor in user errors in handling a multi-layer secured system and attacks against a poorly protected system as seen in real life, the curve would resemble the relation depicted in Figure 3. For security testers, that is of significant importance when catering to implicit security requirements.

Region 1 of the graph depicts low security and low usability. The low usability factor arises from frequent security breaches and unavailability of service due to application downtime for patch release and fixing.

Region 2 of the graph depicts a sharp decline in usability as security is further fortified. Following are the chief reasons:

- Performance degradation due to data encryption and processing overhead
- Difficulty in managing complex keywords with soft keypads or small form factor keypads
- Increase in user errors due to stringent security measures
- Loss of intuitive UI experience due to security countermeasures

To conclude, the Ideal Security region is the most favorable zone. Trading off security is sometimes acceptable if the impact of the risk is minimal compared to the enhancement of usability.

Figure 3: Usability vs. Security Trade-off

Usability vs. Security guidelines:

- Security counter measures should have minimal impact on application performance or user experience.
- Implementing security counter measures should not increase the size of the executable application package significantly.
- The security counter measures should have minimal impact on user-input vectors.
b. Information Gathering Approach for Mobile Platform

As with security testing on any platform, the information gathering phase in mobile-application security testing is the most significant. The data acquired is instrumental for formulating a test plan, test strategy, and penetration testing inputs.

Application Footprint Analysis for Native and Hybrid Applications

Application footprint analysis is aimed at identifying the following goals:

- The data written on device memory at the time of installation
- The data getting written/altered during various transactions
- The residue data after application uninstall

Why Gather Information?

To formulate a test plan, test strategy, and penetration testing inputs.

I. Application Footprint Analysis

<table>
<thead>
<tr>
<th>Step 1:</th>
<th>Pre-install directory structure listing</th>
<th>Before installing the application, a recursive file and folder profile of the device memory should be recorded.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Step 2:</td>
<td>Using hashing functions to create hashes for files and folders</td>
<td>Content hashes such as MD5, SHA1, and SHA 256 can be generated for the files and folders discovered in step 1 and recorded for future use.</td>
</tr>
<tr>
<td>Parameters for the file system to be noted:</td>
<td></td>
<td>Tools such as md5deep are helpful for determining the content hash of the file and folder structure. It is advisable to write the hash value against each directory and file.</td>
</tr>
<tr>
<td>Step 3:</td>
<td>Post-installation directory structure listing and content hash generation</td>
<td>Post-installation recording of directory structure is repeated in a similar fashion as in steps 1 and 2. Record the observation onto a separate file (it pays to have the directory listing in a sorted order by “time last updated”).</td>
</tr>
<tr>
<td>Step 4:</td>
<td>Using diff tools to find difference of content hashes and new files</td>
<td>Use any available diff tool to find the differences between device memory profiles before and after the application installation. This should give a clear picture of which files and directories were created and modified during the installation.</td>
</tr>
<tr>
<td>Step 5:</td>
<td>Perform various transactions in the application and re-hash the files</td>
<td>This is a “deduction through elimination” methodology. Make a content hash of all the concerned files and perform one transaction. Then compare the hashes generated after the transaction. This should reveal all the files that get created and modified during a certain transaction, hence narrowing down the target for penetration testing for information disclosure.</td>
</tr>
<tr>
<td>Step 6:</td>
<td>Carry out uninstallation of application and record directory structure and content hashes.</td>
<td>Run content hash program for all the files and directories similar to step 1 and 2 after uninstallation. Compare the hashes to the pre-install hashes. This should give us all the residue data after uninstallation. This data is essential for further investigations, such as searching for confidential data after a clean uninstallation.</td>
</tr>
</tbody>
</table>
c. Adaptation of Existing Tools and Techniques for Penetration Testing

Limitations in the configurability of devices, and restricted access to file systems and resource files, make the configuration for penetration testing challenging. It is costly, requiring specific proprietary tools. The solution involves adapting existing tools and techniques, thereby reducing the overall testing cost and leveraging existing skills on these tools.

I. Interception of Data Through Proxy

Interception of data for mobile devices is more significant for native and hybrid applications.

There are 3 ways to archive data interception through proxy:

<table>
<thead>
<tr>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuring mobile devices / simulator / emulator to use a proxy</td>
</tr>
<tr>
<td>Configuring a custom network map for mobile data interception</td>
</tr>
<tr>
<td>Configuring mobile device to intercept GPRS and 3G data</td>
</tr>
</tbody>
</table>

Checks can be performed once the data in transmission is available through configured proxy penetration-testing techniques such as data fuzzing, cryptanalysis attacks, and data integrity.

Once the data is intercepted and captured, the system behaves exactly like a client-server system, enabling implementation of penetration-testing techniques involving “man in the middle” attacks.

Modifying web crawlers to mimic mobile browsers is about familiarizing webserver with the browser – doable with Burp suit spider.

II. User Agent Configurable Crawlers

Most of the mobile web application sites are configured for mobile browsers. Possibly, web spiders and crawlers will skip some pages because the web server will refuse connections from clients. It is essential to modify the web crawlers so that they mimic the mobile browsers. The easiest way to do that is to configure the web crawler to send user-agent header as a mobile browser. In Figure 4 (b), we have configured the Burp suit to crawl with Android default browser user agent. Similar mobile browser user-agent strings can be obtained from the web.

Figure 4 (b): Configuring the Burp Suit to Crawl with Android Default Browser User Agent
III. Parsing Application Logs

The application log of any mobile app is the best place to look for security loopholes. All the debug time vestigial code and explanatory logs exist there. That is possible for an abundance of mobile applications while system-wide logs do exist. For a comprehensive study, one can always connect the device to an SDK to retrieve its log files. Tools suiting this process are easily available.

IV. Client Side Security Testing

As the security of a mobile application is fairly dependent on device security, it is essential to conduct a thorough client-side device-level security test. Extensive configuration is not required for that. However, choosing the target device is a challenge. Testers can explore the topic in detail in a later section, titled “Target Device Selection”.

Checklist for client-side security testing:

<table>
<thead>
<tr>
<th>Use Cases</th>
<th>Security Checklist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensitive data is visible/transmittable through an application</td>
<td>The application should enforce device-locking policy</td>
</tr>
<tr>
<td>without authentication</td>
<td></td>
</tr>
<tr>
<td>Sensitive data is written to memory for a transaction</td>
<td>The application should encrypt all sensitive data before committing it to memory</td>
</tr>
<tr>
<td>Sensitive data is transmitted through network</td>
<td>The application should enforce secure network connection for the application to launch</td>
</tr>
<tr>
<td>Configuration files essential for application function</td>
<td>All configuration files should be saved on the device memory</td>
</tr>
<tr>
<td>Application connected to the designated or authorized API</td>
<td>Application should be rooting/jail-breaking aware</td>
</tr>
<tr>
<td>Application uses hardware sensors such as camera, Bluetooth, NFC, or GPS</td>
<td>Application should handle device failures comfortably</td>
</tr>
<tr>
<td>Application takes in sensitive user inputs</td>
<td>Application should not write sensitive user inputs to log, even in case of application failure</td>
</tr>
</tbody>
</table>

**d. Ensuring Test Completeness**

An ideal process is efficient, repeatable, and measurable. Security testing techniques will not suffice for certified application security unless a test completeness framework is in place.

The framework consists of the following 5 verification and validation documents for application security testing:

- Security requirements base lining
- Asset classification
- Threat modelling
- Test cases traceability matrix
- Test execution completeness reporting

**Security Requirement Baselining**

Explicit and implicit security requirements, which serve the scope of security testing, should be documented and baselined strictly before all security testing activities.

**Asset Classification**

The documentation should contain all the data & resources being used by the application. From that list, sensitive and non-sensitive data are segregated and put into order in a criticality-index file. Data having 5 as criticality index is considered highly sensitive, and 0 as non-sensitive. This document needs to be signed and agreed upon before security goals are prepared.
Threat Modeling

The goal should be to protect mission-critical data without impeding usability or performance of the application. While many threat modelling paradigms exist in the industry, past experiences rank “DREAD” as the best suited model for mobile application security testing. In short, DREAD stands for parameters used to evaluate a risk. They are: Damage, Reproducibility, Exploitability, Affected Users, and Discoverability.

A table to illustrate transformation of the DREAD model for the mobile platform:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Evaluation Criteria</th>
<th>Scoring</th>
</tr>
</thead>
</table>
| Damage          | The criticality of the data being compromised by the threat                          | › Minor loss of confidentiality/integrity: 0 to 4 pts.  
|                 |                                                                                     | › Moderately sensitive data: 5 to 7 pts.       
|                 |                                                                                     | › Critical data: 8 to 10 pts.                 |
| Damage          | The criticality of the data being compromised by the threat                          | › Minor loss of confidentiality/integrity: 0 to 4 pts.  
|                 |                                                                                     | › Moderately sensitive data: 5 to 7 pts.       
|                 |                                                                                     | › Critical data: 8 to 10 pts.                 |
| Reproducibility | The degree of difficulty in reproducing the security attack                          | › Extremely difficult: 0 to 4 pts.             
|                 |                                                                                     | › Moderately difficult: 5 to 7 pts.            
|                 |                                                                                     | › Easy: 8 to 10 pts.                           |
| Exploitability  | The degree of difficulty in extracting data from plain text                          | › Extremely difficult: 0 to 4 pts.             
|                 |                                                                                     | › Moderately difficult: 5 to 7 pts.            
|                 |                                                                                     | › Easy: 8 to 10 pts.                           |
| Affected users  | Threats impacting affected users and vice versa (across devices), by affected-users index | › Highly device/platform specific: 0 to 4 pts.  
|                 |                                                                                     | › Moderately device/platform specific: 5 to 7 pts.  
|                 |                                                                                     | › System-wide vulnerability: 8 to 10 pts.      |
| Discoverability | Discoverability index is a measure of the difficulty in identifying the threat      | › Needs application source code or server-side admin privilege: 0 to 4 pts.  
|                 |                                                                                     | › Needs admin privilege over mobile devices (rooted/jail broken): 5 to 7 pts.  
|                 |                                                                                     | › No customization required: 8 to 10 pts.       |

\[
\text{DREAD Index} = \text{Damage} \times \text{Reproducibility} \times \text{Exploitability} \times \text{Affected Users} \times \text{Discoverability}
\]
**Test Cases Traceability Matrix**

The security test case traceability matrix is a measure of test coverage. The document, based on the security requirements, ensures that all test cases are in place. For every test case, there must be a threat associated with a DREAD index of 4 or more. The threat should affect an asset with criticality index of 2 or more. The asset should occur in the use cases mentioned in the requirement specifications. To summarise the traceability matrix of test cases, documents will contain the relation illustrated in Figure 5.

<table>
<thead>
<tr>
<th>Security Requirement</th>
<th>Assets &amp; Criticality Index</th>
<th>Threat &amp; DREAD Index</th>
<th>Testcase</th>
</tr>
</thead>
</table>

![Figure 5: Components of Security Test Case Traceability](image)

**Test Execution Completeness Reporting**

Test execution completeness should be bound by a defined exit criterion. Such criteria should determine whether to graduate a pre-production build into production. Traditionally, one would have used the set of high-severity test cases as gate criteria.

- No failure of test cases corresponding to threats having a DREAD index of 6 or more
- 100% test case execution coverage is ensured
- Application is stable and no other release is planned

**For mobile application security testing, here are the test completeness criteria:**

- No failure of test cases corresponding to threats having a DREAD index of 6 or more
- 100% test case execution coverage is ensured
- Application is stable and no other release is planned

**2. Knowledge Base Management (Paranoid Beehive policy)**

The efficiency of a process depends on the maturity of the execution model and the collective experience of the team implementing it. Improvement of reporting structures helps better quantify outcomes of a process. However, effective counter measures may not be easy to achieve unless the expertise is updated to meet requirements in a fresh scenario.

- Formation of security test team forum
  - Special interest groups and test engineering forums provide the ideal platform on which knowledge base management can be implemented
  - These test engineering forums should be self-driven and self-sustaining R&D units within the team
  - Should the team size permit, each member of the team should be given the ownership of one branch of mobile security testing, such as Android file system management, mobile browser emulators, mobile device simulators, cryptanalysis, etc.
- Documentation of all known threat summary reports (format)
  - Most of the new attacks and threats are often derivatives of known threats and vulnerabilities. It is beneficial to document all known threats in order to better understand unknown threats and their attack methodologies.
  - The document should highlight: Threat, Impact, Category of applications affected, and Tools to verify
  - Provide professionals an extract of the recent list of threat summary reports as a starting point
- Regular publishing of bulletins on new technology in mobile platform is highly advisable (Ex: NFC payment systems, QR code reader & biometric recognition systems, etc.)
- Collective brainstorming sessions to pre-empt risks and threats corresponding to new technology; i.e. ensuring that a new threat makes it to the threat summary report
- Research based on other sources such as OWASP top 10, PCI DSS guidelines, etc. for identifying new frontiers in application security
- Periodic review of application-store security policies and requirements such as Google Play, Apple App Store, etc. to stay aware of security and quality policies which change intermittently
### Threat Summary Report

<table>
<thead>
<tr>
<th>Mobile Risks (TOP 5)</th>
<th>Impact</th>
<th>Applicable apps</th>
<th>Tools</th>
</tr>
</thead>
</table>
| Weak Server Side Control             | *Confidentiality of data lost  
  *Integrity of data not trusted | Web apps, Hybrid apps                          | iGoat V2.0    |
| Insecure Data Storage                | *Confidentiality of data lost  
  *Credentials disclosed  
  *Privacy violations         | Native apps, Web apps, Hybrid apps             | iGoat V2.0    |
| Insufficient Transport Layer Protection | *Man-in-the-middle attack  
  *Tampering data in transit  
  *Confidentiality of data lost | Web apps, Hybrid apps                          | Wireshark, Mobisec |
| Side Channel Data Leakage            | *Data retained indefinitely  
  *Privacy violations         | Native apps, Web apps, Hybrid apps             | iGoat V2.0    |
| Poor Authorization and Authentication | *Privilege escalation  
  *Unauthorized access         | Native apps, Web apps, Hybrid apps             | ProxyDroid (BurpSuit) |

### 3. Target Device Selection

It is essential to select the correct device to be used as the platform under testing. Without due diligence in the process, security testing will fail to uncover flaws that arise from platform-specific parameters such as limited physical memory, custom ROM routines, unavailability of required sensors (camera, gyroscope, NFC, Bluetooth, etc.).

#### 3 classes of devices are essential in a test lab used for security testing:

<table>
<thead>
<tr>
<th>Whitelist devices</th>
<th>Grey-list devices</th>
<th>Blacklist devices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Those which are fully supported and recommended for the application under test. (For example, an application developed for iOS 6, the iPhone 5 is considered as a whitelist device.)</td>
<td>Those which are partially supported or at least expected to be supported due to backward compatibility feature of SDK. (For example, an iPod 4th generation device would be considered a grey-list device for our previous example.)</td>
<td>Those, for which the applications were not targeted during development. In most cases, jail-broken or rooted devices serve as blacklist devices. It is the class of devices most likely to uncover security vulnerabilities as the devices deviate from manufacturers’ application security guidelines.</td>
</tr>
</tbody>
</table>

Although it is possible to use simulators and emulators for this purpose, it is not advisable. A simulator will not accurately mimic the limitations of a mobile device, which might lead to numerous false positives in test reports. An organization or management should assign relatively low priority to the popularity of devices when selecting target devices. For mobile web applications, one may either use any cross-browser testing tool available in the market, or opt for physical devices that provide accurate simulation of device-specific limitations (limited processing power, lightweight JavaScript engine, limited network bandwidth).
Conclusion

From an attacker’s point of view, a mobile device is probably the most desirable target. Most mobile devices contain aggregated, highly sensitive, and subjective data about people and organizations. That is the primary reason behind industry-wide urgency in incorporating effective security standards for mobile applications.

Moreover, structured security testing enables significant cost savings and productivity gains, thus contributing to greater ROI and risk mitigation. According to the National Institute of Standards and Technology (NIST), the cost of fixing a vulnerability at a later stage, i.e. after its acceptance, can be 30 times greater compared to the cost of fixing it during the design phase.

While tools are reputed to improve returns greatly, adequate security is always advisable. It may be difficult to measure the returns on a security investment directly. However, definite outcomes include greater reliability, lower expenditure, and long-term improvement in ROI. In fact, a combination of high-level analysis, low-level review, metrics-based risk management, and associated tools can help measure ROI.

The primary philosophy behind this white paper is to encourage a culture of improvement in security processes, as it improves organizational maturity over a period. Branching out from web security testing, the processes in this white paper should enable test professionals to identify, quantify, and ultimately mitigate security threats better.

In brief:

- Through increased security, it is possible to acquire a certain degree of trust among end users.
- Employing security testing in a structured approach produces an efficient and superior ROI, which cannot be directly measured, but can help to mitigate or avoid potential risks.
- A study by the National Institute for Standards and Technology (NIST) estimated that the cost of fixing a vulnerability that was not discovered until acceptance testing might be up to 30 times greater compared to fixing it in the design phase.

About the Authors

Bhabani Prasad Swain serves as a Sr. Test Engineer at Tavant Technologies. Specializing in Mobile Platforms Security Testing, he is currently part of the Non-functional Requirement Test Team at Tavant Technologies. He is a Certified Ethical Hacker and holds B. Tech in Electronics and Telecommunication. He has recently been involved in restructuring the security test team at Tavant Technologies with his “Paranoid Beehive” policy.

Rahul Kumar Sinha is a Quality Assurance Lead at Tavant Technologies. He has extensive experience in Performance, Security, Localization, and Compatibility testing. Being involved in creating performance & security test plans, estimation plans and automation scripts, he is currently involved in improving the techniques and methodologies used in performance and security testing of mobile applications.

Keshava Murthy is part of Tavant Technologies in the capacity of Quality Assurance Manager. He has a total industry experience of more than 10 years. He has been mentoring test engineers in some of the significant areas of technology, such as performance testing of various platforms like web service, mobile, and e-business applications. He is a subject matter expert in Non-functional Requirement testing which includes, Performance, Security, Globalization, and Compatibility testing.
References

Agile methodologies: http://scrumtrainingseries.com/


Wire shark documentation: http://www.wireshark.org/docs/wsug_html_chunked/

PCI DSS documentation: https://www.pcisecuritystandards.org/security_standards/documents.php?document=pci_dss_v2-0#pci_dss_v2-0

http://www.nngroup.com/articles/mobile-native-apps/
## Appendix

<table>
<thead>
<tr>
<th>Terms</th>
<th>Definitions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decomposition Application</td>
<td>The process of logically fragmenting various modules in an application to better understand the control flow and function of an application.</td>
</tr>
<tr>
<td>Base Lining</td>
<td>A set of configurations, requirements, or use cases serving as a basis of further measurements or tests.</td>
</tr>
<tr>
<td>Consumer Mobile Application</td>
<td>Mobile application aimed at personal use.</td>
</tr>
<tr>
<td>Cryptanalysis</td>
<td>The science of finding weaknesses in cryptographic algorithms.</td>
</tr>
<tr>
<td>Data Integrity</td>
<td>The process of maintaining accuracy and completeness of data.</td>
</tr>
<tr>
<td>Enterprise Mobile Applications</td>
<td>Special purpose application containing proprietary data, logic, or derivative thereof.</td>
</tr>
<tr>
<td>False Positive</td>
<td>A threat detected in test phase, which is inconsequential in the real world, such as a cryptanalysis estimate of 273 years of 100 Tera FLOPS for an encryption algorithm.</td>
</tr>
<tr>
<td>FLOPS</td>
<td>Floating-point Operations Per Second is a measure of computer performance.</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System sensor.</td>
</tr>
<tr>
<td>Human Interface Capabilities</td>
<td>The facility of a device through which users interact. In mobile devices, these consist of the touch screen, microphone, hardware keys, sensors such as the proximity sensor, and gyroscope, etc.</td>
</tr>
<tr>
<td>Hybrid Application</td>
<td>The class of mobile applications having a client application for fast user responsiveness and a server application for heavy processing.</td>
</tr>
<tr>
<td>Learning Curve</td>
<td>The measure of operational efficiency with respect to time spent on research, learning, and training.</td>
</tr>
<tr>
<td>Mobile Web Application</td>
<td>The class of web pages specially engineered to be used in low resolution and touch-enabled mobile devices.</td>
</tr>
<tr>
<td>Native Application</td>
<td>The class of mobile applications having only client-side functionality. It is considered a light-weight application with very little or no connectivity to any server.</td>
</tr>
<tr>
<td>NFC</td>
<td>Near Field Communication technology.</td>
</tr>
<tr>
<td>PCIDSS</td>
<td>Payment Card Industry Data Security Standard.</td>
</tr>
<tr>
<td>OWASP</td>
<td>Open Web Application Security Project.</td>
</tr>
<tr>
<td>Penetration Testing</td>
<td>The process of simulating an attack in a controlled environment to determine the severity of known threats or security defects.</td>
</tr>
<tr>
<td>SDLC</td>
<td>Software Development Lifecycle.</td>
</tr>
<tr>
<td>Target Device/Platform</td>
<td>The devices/platform under test in a security testing process.</td>
</tr>
</tbody>
</table>