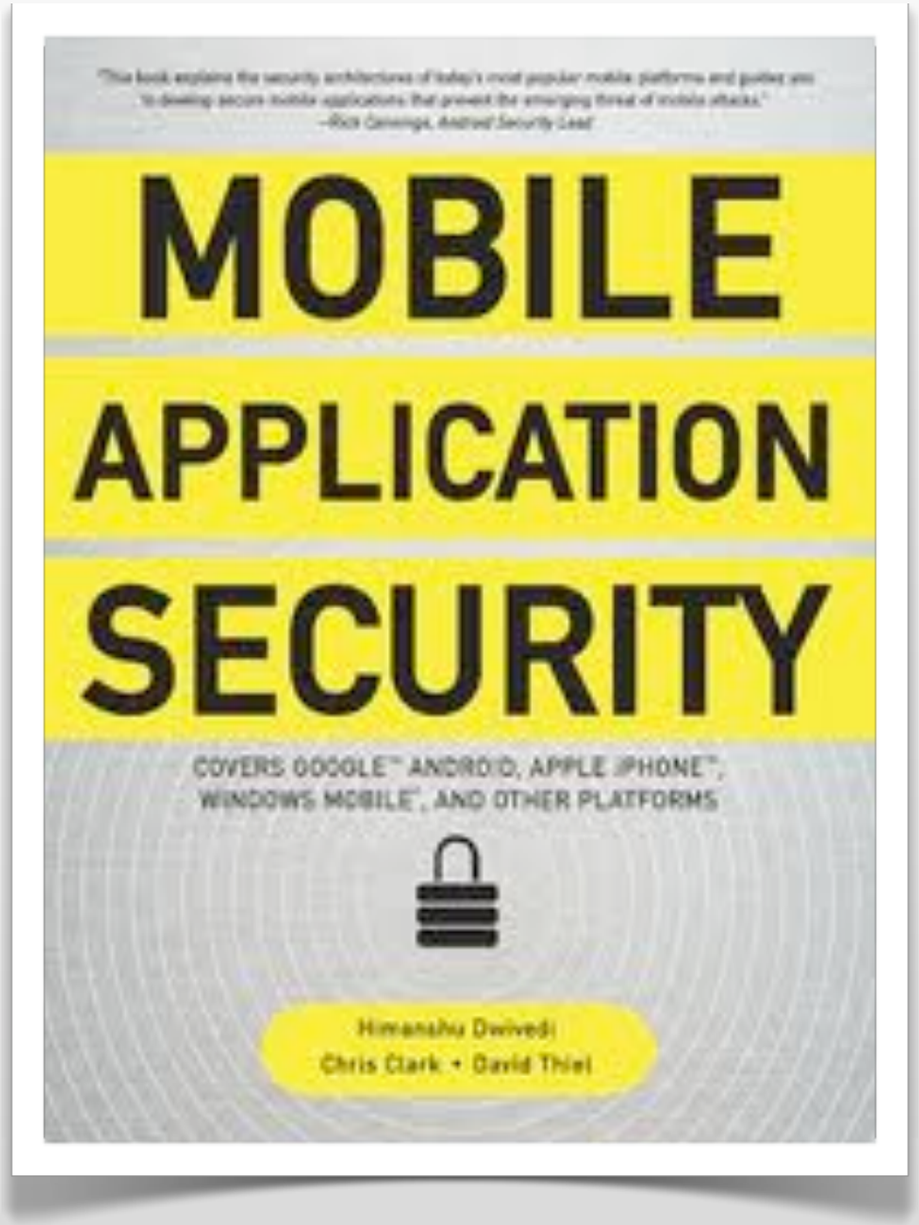
EECS 710 - Fall 2012

Mobile Application Security

Himanshu Dwivedi
Chris Clark
David Thiel

Presented by
Bharath Padmanabhan



The Authors

Himanshu Dwivedi is a co-founder of iSEC Partners, an information security firm specializing in application security. He is also a renowned industry author with six security books published.

Chris Clark is a principal security consultant at iSEC Partners, where he writes tools, performs penetration tests, and serves as a Windows and Mobile expert.

David Thiel is a Principal Security Consultant with iSEC Partners. His areas of expertise are web application penetration testing, network protocols, fuzzing, Unix, and Mac OS X.

Contents

Part I Mobile Platforms

Top Issues Facing Mobile Devices

- Physical Security
- Secure Data Storage (on Disk)
- Strong Authentication with Poor Keyboards
- Multiple-User Support with Security
- Safe Browsing Environment
- Secure Operating Systems
- Application Isolation
- Information Disclosure
- Virus, Worms, Trojans, Spyware, and Malware
- Difficult Patching/Update Process
- Strict Use and Enforcement of SSL
- Phishing
- Cross-Site Request Forgery (CSRF)
- Location Privacy/Security
- Insecure Device Drivers
- Multifactor Authentication

Top Issues Facing Mobile Devices

Physical Security

- Loss of information from lost or stolen devices
- Unauthorized usage by borrower
- Physical security has always meant little-to-no security

Secure Data Storage (on Disk)

- Sensitive information stored locally (password files, tokens, etc.)
- Prevent unauthorized access while making it accessible to certain applications on an as-needed basis

Top Issues Facing Mobile Devices

Strong Authentication with Poor Keywords

- Password or passphrase that uses a combination of letters, numbers, special characters, and a space
- Same standard on a mobile keyboard is difficult, if not impossible

Multiple-User Support with Security

- Unlike traditional client operating systems that support multiple users with different operating environments, no such thing as logging into a mobile device as a separate user
- No distinction between applications for business purpose vs. personal
- Need unique security model by application to prevent data exposure

Top Issues Facing Mobile Devices

Safe Browsing Environment

- Lack of real estate makes phishing attempts easier
- Inability to view the entire URL or the URL at all
- Links are followed a lot more on mobile devices

Secure Operating Systems

- Securing an OS is no easy task but should still be undertaken by all mobile vendors
- Security often correlates to data loss but can also correlate to system downtime and diminished user experience

Top Issues Facing Mobile Devices

Application Isolation

- Very common to see various types of applications (corporate, gaming, social, etc) on a mobile device
- Ability to isolate these applications and the data they require is critical

Information Disclosure

- Data stored on a device (desktop, laptop, server, mobile) is worth more than the device itself, however, mobile device more likely to be lost or stolen
- Access from mobile device to other networks (say VPN) is another area of concern if authentication mechanisms are not strong

Top Issues Facing Mobile Devices

Virus, Worms, Trojans, Spyware, and Malware

- Mobile devices also face threat of viruses, worms, Trojans, spyware, and malware
- Lessons to learn from the desktop world but also need to adjust to the mobile environment and new attack classes

Difficult Patching/Update Process

- Patching and updating not a technical challenge but several considerations make it a difficult problem for mobile
- Carriers have big problems with immediate system updates and patching due to little response time for testing
- Requires coordination among OS developer, carriers, and handset vendors

Top Issues Facing Mobile Devices

String Use and Enforcement of SSL

- Older devices lacked horsepower to enforce SSL without affecting user experience; some still allowed for backwards-compatibility
- Some organizations defaulting to clear-text protocols assuming increased complexity of sniffing on 3G network
- Abundance of transitive networks between mobile device and the end system

Phishing

- Users more prone to clicking links on mobile without safety concerns
- Lack of real estate to show entire URL or the URL itself

Top Issues Facing Mobile Devices

Cross-Site Request Forgery (CSRF)

- Big problem for mobile HTML sites that are vulnerable
- Easy to get victims to click on links due to previously mentioned factors
- Allows attacker to update a victim's information (address, email, password, etc) on a vulnerable application

Location Privacy/Security

- Most mobile users have assumed their location privacy was lost as soon as they started carrying a mobile device
- Users willingly give away their location-specific information through applications like Google Latitude, Foursquare etc.

Top Issues Facing Mobile Devices

Insecure Device Drivers

- Most applications should not have system access to mobile device but device drivers need such access
- Exposure to attackers if third-party drivers provide methods to get around protection schemes via potentially insecure code

Multifactor Authentication

- Soft multifactor authentication schemes (same browser, IP range, HTTP headers) used by mobile web applications very vulnerable to spoofing
- Typical to create a device signature using a combination of HTTP headers and properties of the device's connection but still not good enough compared to native mobile applications

Tips for Secure Mobile Application Development

- Leverage TLS/SSL
- Follow Secure Programming Practices
- Validate Input
- Leverage the Permissions Model Used by the OS
- Use the Least Privilege Model for System Access
- Store Sensitive Information Properly
- Sign the Application's Code
- Figure out a Secure and Strong Update Process
- Understand the Mobile Browser's Security Strengths and Limitations
- Zero Out the Non-Threats
- User Secure/Intuitive Mobile URLs

Tips for Secure Mobile Application Development

Leverage TLS/SSL

- Turn on Transport Layer Security (TLS) or Secure Sockets Layer (SSL) by default
- Both confidentiality and integrity protections should be enabled; many environments often enforce confidentiality, but do not correctly enforce integrity protection

Follow Secure Programming Practices

- Big rush (and a small budget) to get a product out the door, forcing developers to write code quickly and not make the necessary security checks and balances
- Leverage the abundance of security frameworks and coding guidelines available

Tips for Secure Mobile Application Development

Validate Input

- Validating input is imperative for both native mobile applications and mobile web applications
- Mobile devices do not have host-based firewalls, IDS, or antivirus software, so basic sanitization of input is a must

Leverage the Permissions Model Used by the OS

- Permissions model is fairly strong on the base device, however, external SD card may not be as secure
- Application isolation provided by systems like iOS and Android should be leveraged

Tips for Secure Mobile Application Development

Use the Least Privilege Model for System Access

- The least privilege model involves only asking for what is needed by the application
- One should enumerate the least amount of services, permissions, files, and processes the application will need and limit the application to only those items
- The least privilege model ensures the application does not affect others and is run in the safest way possible

Store Sensitive Information Properly

- Do not store sensitive information (usernames, passwords, etc.) in clear text on the device; use native encryption schemes instead

Tips for Secure Mobile Application Development

Sign the Application's Code

- Although signing the code does not make the code more secure, it allows users to know that an application has followed the practices required by the device's application store
- Unsigned application may have a much reduced number of privileges on the system and will be unable to be widely disturbed through the various application channels of the devices
- Depending on whether or not the application is signed, or what type of certificate is used, the application will be given different privileges on the OS

Tips for Secure Mobile Application Development

Figure Out a Secure and Strong Update Process

- Much like in the desktop world, an application that is not fully patched is a big problem for the application, the underlying OS, and the user
- A secure update process needs to be figured out where an application can be updated quickly, easily, and without a lot of bandwidth

Understand the Mobile Browser's Security Strengths and Limitations

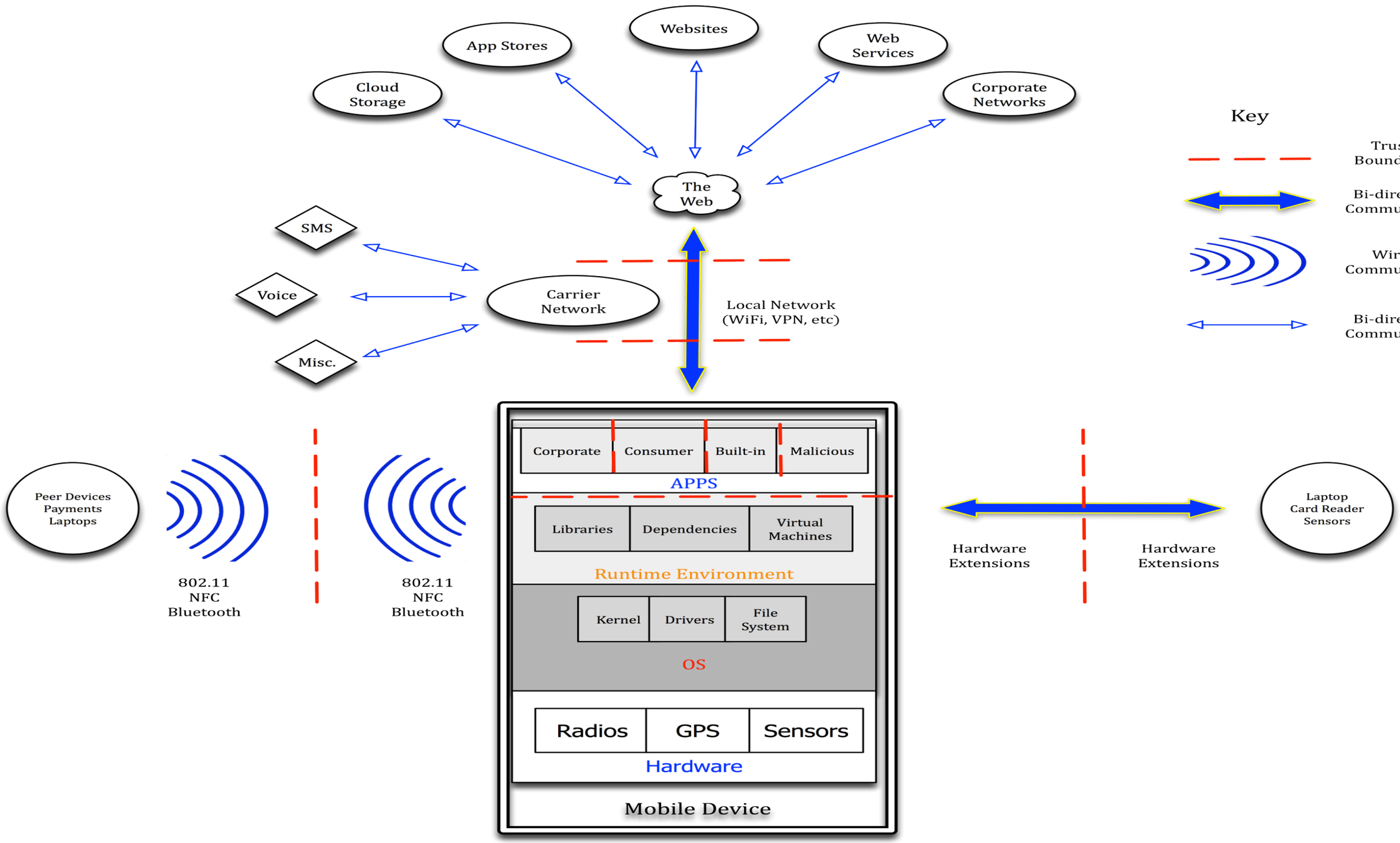
- Understand the limitations of cookies, caching pages locally to the page, the Remember Password check boxes, and cached credentials
- Do not treat the mobile browser as you would treat a regular web browser on a desktop operating system

Tips for Secure Mobile Application Development

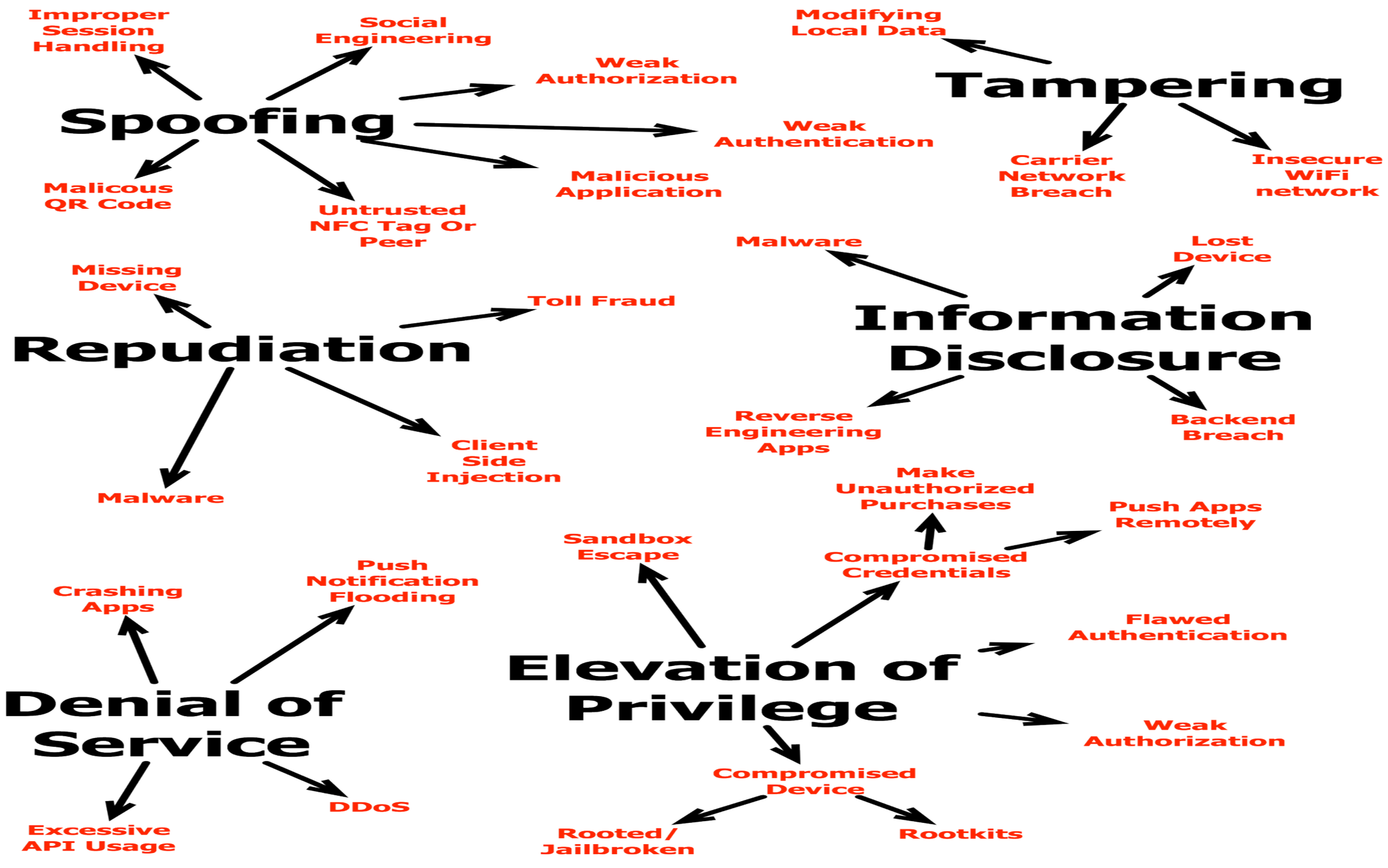
Zero Out the Non-Threats

- Although the threats to mobile devices and their applications are very real, it is important to understand which ones matter to a given application
- The best way to start this process is to enumerate the threats that are real, design mitigation strategies around them, and note the others as accepted risks ("threat model")
- Threat model should allow application developers to understand all the threats to the system and enable them to take action on those that are too risky to accept

OWASP Mobile Threat Model



OWASP Mobile Threat Model



Tips for Secure Mobile Application Development

Use Secure/Intuitive Mobile URLs

- Some organizations use third-parties to host their mobile sites whose domain will be different from the organization
- Many organizations have mobile-optimized sites separate from their regular websites but it is important to keep the URLs intuitive

Intuitive

- m.isecpartners.com

Not So Intuitive

- isecpartners.mobi
- isecpartners.mobilevendor.com

Apple iPhone

- Development
- Security Testing
- Application Format
- Permissions and User Controls
- Local Data Storage
- Networking
- Push Notifications, Copy/Paste, and Other IPC

Development

- Performed with Xcode and the iPhone SDK
- Can be run either within the emulator or on a physical device
- Debugging is done within Xcode via gdb
- Objective-C code can be decompiled fairly easily using standard OS X developer tools
- It is *not* possible to prevent reverse-engineering of the code

Security Testing

- Threat of classic C exploits is reduced, but not eliminated, by using high-level Objective-C APIs
- To avoid buffer overflows, avoid manual memory management and use Cocoa objects such as NSString for string manipulation (Integer overflows still possible even when using NSInteger)
- Double-frees, where a segment of memory is already freed from use and an attempt is made to deallocate it again, are a problem
- Most commercial static analysis tools haven't matured to detect Objective-C specific flaws, but simple free tools can be used to fine C API abuses

Application Format

- Applications are compiled via Xcode using the GNU GCC compiler, cross-compiled for the ARM processor and local machine (emulator)
- Each application bundle includes a unique ID, a plist of entitlements and preferences, a code signature, any required media assets, and the executable itself
- All iPhone applications have to be distributed through the “App Store” which have to be approved by Apple prior to distribution and can be revoked anytime at Apple’s discretion
- All iPhone applications have to be signed by a valid code-signing certificate; requires membership with the iPhone Developer Program
- On “jailbroken” iPhones, using Cydia and Installer are the two most popular ways to install unauthorized third-party software

Permissions and User Controls

- Apple uses “Mandatory Access Control” (MAC) as its mechanism for restricting the capabilities of applications
- The iPhone OS and OS X permission system (“sandboxing”) is based on the TrustedBSD framework which allows for writing policy files that describe what permissions an application should have
- Each application is installed into its own directory (GUID); they are allowed limited read access to some system areas but not allowed to read/write directories belonging to other applications
- Both the heap and stack are non-executable by default; newer versions support ASLR (Address Space Layout Randomization)
- Permissions granting for specific functionality (location, contacts) is granted via popups to the user at the time of API use

Local Data Storage

- **SQLite Storage:** This is a popular way to persist application data but is subject to injection attacks like any type of SQL database. Parameterized queries should be used to ensure third-party SQL is not accidentally executed by the application.

- **Keychain Storage:** The iPhone includes the Keychain mechanism from OS X (with some differences) to store credentials and other data. The API is simpler compared to the Cocoa API and all the data is stored in a dictionary of key/value pairs. It is to be noted that Keychain APIs only work on a physical device.

- **Shared Keychain Storage:** iOS 3.0 introduced this ability allowing for separate applications to share data by defining additional "entitlements". The developer has to explicitly specify this when adding the attribute and also define the entitlement.

Networking

- **URL Loading API:** Supports HTTP, HTTPS, FTP, and file resource types using the NSURLConnection and NSURLDownload APIs with an NSURL object as the input. It is to be noted that HTTP and HTTPS request results are cached on the device by default and all cookies stored are accessible by any application that uses the URL loading system.
- **NSStreams:** Useful when using network sockets for protocols other than those handled by the URL loading system, or in places where you need more control over how connections behave.
- **Peer to Peer (P2P):** iOS 3.0 introduced this ability to do P2P networking between devices via Bluetooth. Opportunities for data theft are increased since game and non-game applications use it for collaboration and data exchange. Also, because data can potentially be streamed to the device by a malicious program or user, it is another untrusted input to be dealt with.

Push Notifications, Copy/Paste, and Other IPC

- **Push Notifications:** iOS 3.0 introduced this ability allowing applications to provide users with notifications when they are not running. The device and push service platform perform mutual certificate authentication. Notification types can include popups or updating the badge. It should be noted that push notifications are *not* guaranteed to be delivered.

- **UIPasteboard:** Similar to OS X, this can be implemented to handle copying and pasting of objects within an application, or to handle data to share among applications. Copied and pasted data is stored in item groupings with various representations. Any information on shared pasteboards should be considered untrusted and potentially malicious and needs to be sanitized before use.

Google Android

- Development
- Platform Security Architecture
- Security Model
- Permissions
- Securable IPC Mechanisms
- Application Signing
- Memory Management Security Enhancements
- Files, Preferences, and Mass Storage

Development

- SDK provides free tools for building and debugging applications, supporting developers on Linux, Windows, and OS X
- SDK provides an emulator that emulates ARM-based device and also alternate virtual hardware configurations
- Debugging support is built into Android and working with a device or with the emulator is mostly interchangeable
- Code developed using the SDK generally runs in the Dalvik VM

Platform Security Architecture

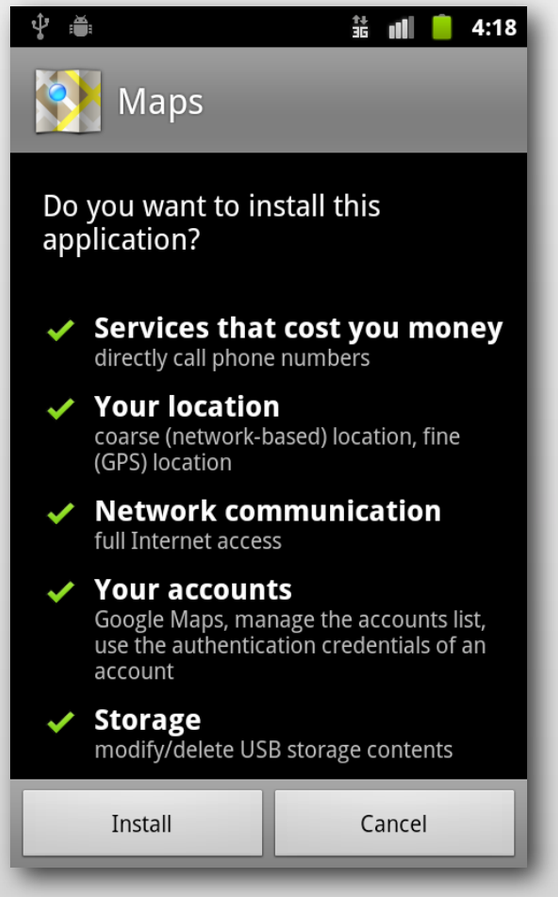
- Android seeks to be the most secure and usable operating system for mobile platforms by re-purposing traditional operating system security controls to:
  - Protect user data
  - Protect system resources (including the network)
  - Provide application isolation
- To achieve these objectives, Android provides these key security features:
  - Robust security at the OS level through the Linux kernel
  - Mandatory application sandbox for all applications
  - Secure interprocess (IPC) communication
  - Application signing
  - Application-defined and user-granted permissions

Security Model

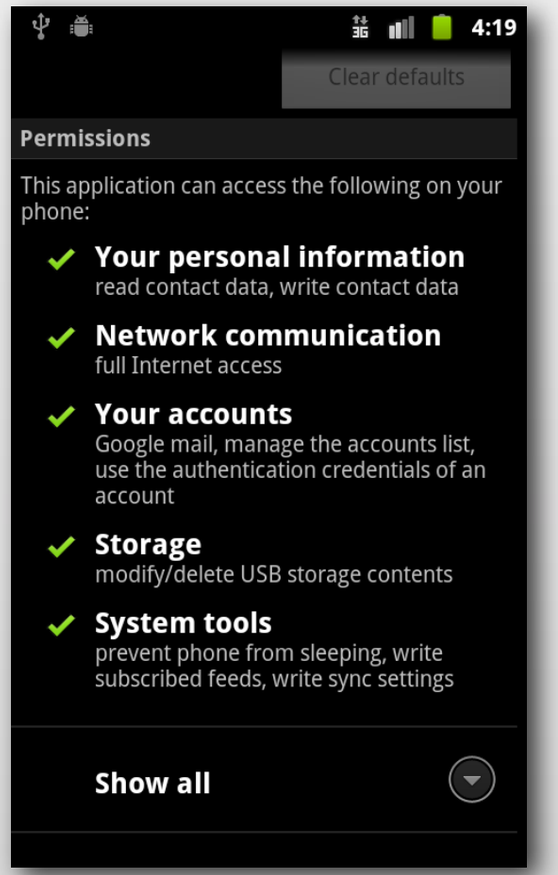
- Android is based on the Linux security model with some abstractions unique to it and leverages Linux user accounts to silo applications
- Android permissions are rights given to applications to allow them to take pictures, use the GPS, make phone calls, and so on
- When installed, applications are given a unique user identifier (UID); the UID is used to protect an application's data
- The need for permissions minimizes the impact of malicious software, unless a user grants powerful rights to dubious software
- Android's runtime system tracks which permissions each application has; these permissions are granted either when the OS was installed or upon installation of the application by the user

Permissions

- Android uses *manifest permissions* to track what the user allows applications to do, such as sending SMS, using the camera, etc.
- Prior to installation of any application, the user is shown the different permissions the application is requesting. Once installed, an application's permissions *cannot* be changed.



Permissions at Application Install



Permissions of an Installed Application

Permission Protection Levels

Protection Levels	Protection Behavior
Normal	Permissions for application features whose consequences are minor (for example, VIBRATE, which lets applications vibrate the device). Suitable for granting rights not generally of keen interest to users. Users can review them but may not be explicitly warned.
Dangerous	Permissions such as WRITE_SETTINGS and SEND_SMS are dangerous because they could be used to reconfigure the device or incur tolls. Use this level to mark permissions users will be interested in or potentially surprised by. Android will warn users about the need for these permissions upon install, although the specific behavior may vary according to the version of Android or the device upon which it is installed.
Signature	These permissions are only granted to other applications signed with the same key as the program. This allows secure coordination without publishing a public interface.
SignatureOrSystem	Similar to Signature, except that programs on the system image also qualify for access. This allows programs on custom Android systems to also get the permission. This protection helps integrate system builds and won't typically be needed by developers. **Note:** Custom system builds can do whatever they like. Indeed, you ask the system when checking permissions, but SignatureOrSystem-level permissions intend for third-party integration and thus protect more stable interfaces than Signature.

Securable IPC Mechanisms

- **Activities** are interactive screens used to communicate with users. Intents are used to specify an Activity.
- **Broadcasts** provide a way to send messages between applications. When sending a broadcast, an application puts the message to be sent into an Intent.
- **Services** are background processes that toil away quietly in the background.
- **ContentProviders** provide a way to efficiently share relational data between processes securely. They are based on SQL.
- **Binder** provides a highly efficient communication mechanism. It is commonly used to bridge Java and native code running in separate processes.

Application Signing

- Every application that is run on the Android platform must be signed by the developer. Applications that attempt to install without being signed will rejected by either Google Play or the package installer on the Android device.
- The signed application certificate defines which user id is associated with which application. Application signing ensures that one application cannot access any other application except through well-defined IPC.
- Applications can be signed by a third-party or self-signed. Android provides code signing using self-signed certificates that developers can generate without external assistance or permission.

Memory Management Security Enhancements

- ProPolice to prevent stack buffer overruns
- safe_iop to reduce integer overflows
- Extensions to OpenBSD dlmalloc to prevent double free() vulnerabilities and to prevent chunk consolidation attacks
- OpenBSD calloc to prevent integer overflows during memory allocation
- Format string vulnerability protections
- Hardware-based No eXecute (NX) to prevent code execution on the stack and heap
- Linux mmap_min_addr to mitigate null pointer dereference privilege escalation
- Address Space Layout Randomization (ASLR) to randomize key locations in memory

Files, Preferences, and Mass Storage

- UNIX-style file permissions are present in Android; each application runs as its own user so files created by one application cannot be read or altered by another application (unless user allows it)
- SharedPreferences is a system feature that is backed by a file with permissions like any others
- Android devices may support larger add-on file systems mounted on memory cards since devices typically have limited amount of memory
- Data stored on memory cards is unprotected and cannot be accessed by any program on the device

Contents

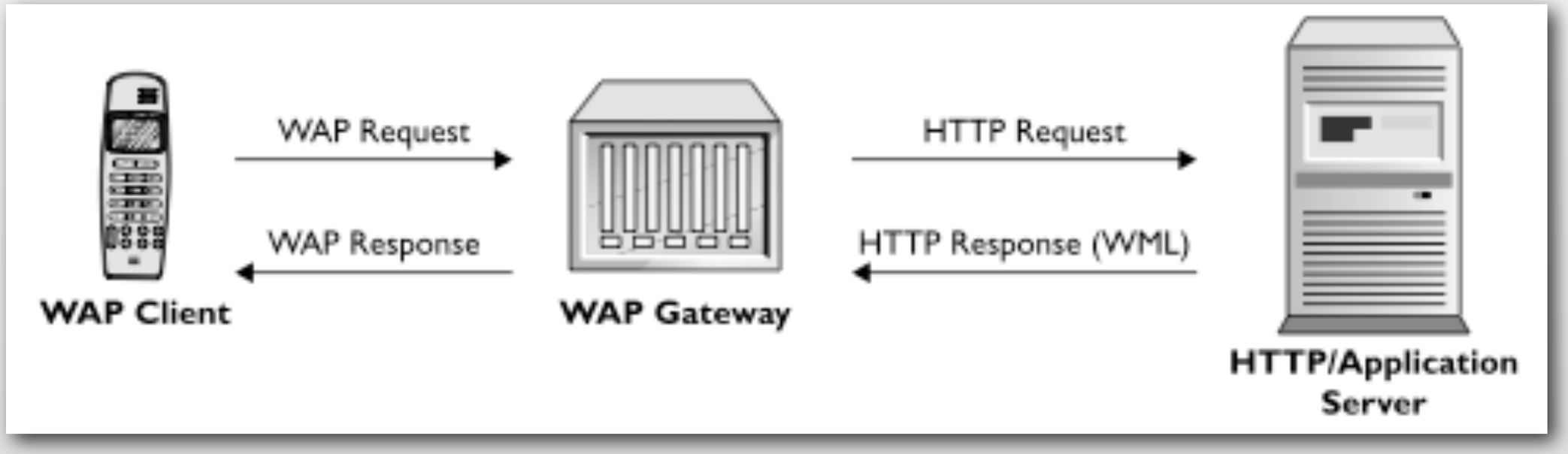
Part II Mobile Services

WAP and Mobile HTML Security

- WAP and Mobile HTML Basics
- Authentication on WAP/Mobile HTML Sites
- Encryption
- Application Attacks on Mobile HTML Sites
- WAP and Mobile Browser Weaknesses

WAP and Mobile HTML Basics

- WAP is a method to access the Internet from mobile devices
- WAP gateway acts like a proxy server translating content
- WAP 2.0 does not require a WAP gateway



WAP Architecture

Authentication on WAP/Mobile HTML Sites

- One of the many problems that WAP and Mobile HTML developers have with mobile devices is the keyboard.

PDA-style keyboard

Non-PDA-style keyboard

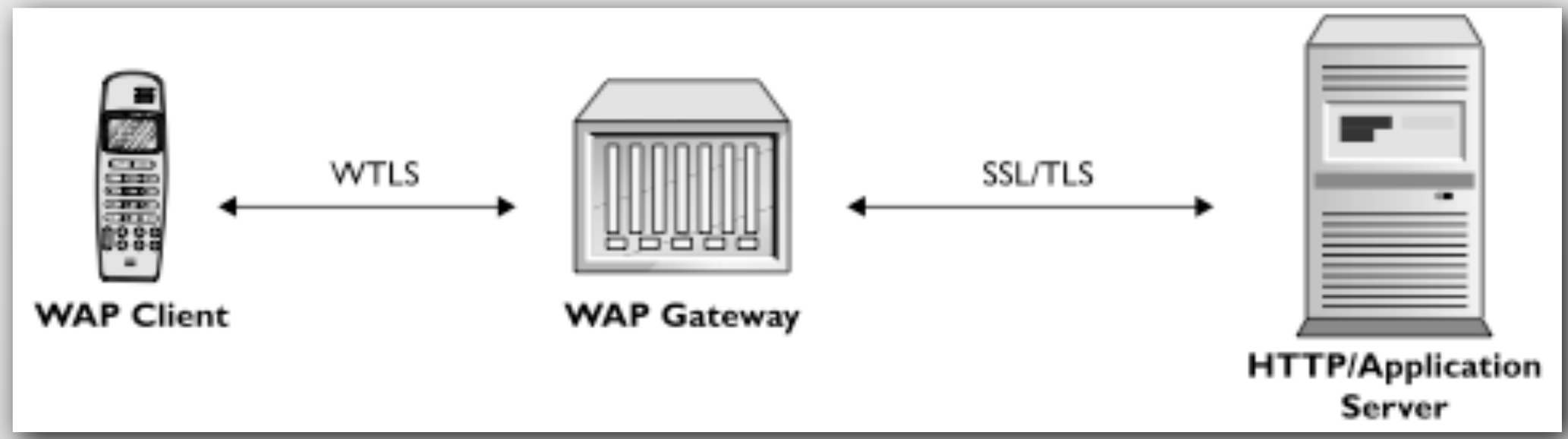
- Strong passwords (consisting of letters, numbers, and special characters) difficult to type on Non-PDA-style keyboards.

Authentication on WAP/Mobile HTML Sites

- Mobile PIN, an alternative to complex passwords, increases the user experience at the cost of lowering the security of the authentication process
- PIN typically consists of only 4-8 digit numbers, making it easier for brute-force attempts
- Crossover use of SMS and WAP/Mobile HTML applications is another avenue of exposure. Ex: Sending an SMS message to a predefined number will return an account balance as long as the caller ID value is correct.
- Spoofing of caller ID (“trusted value”) is quite simple

Encryption

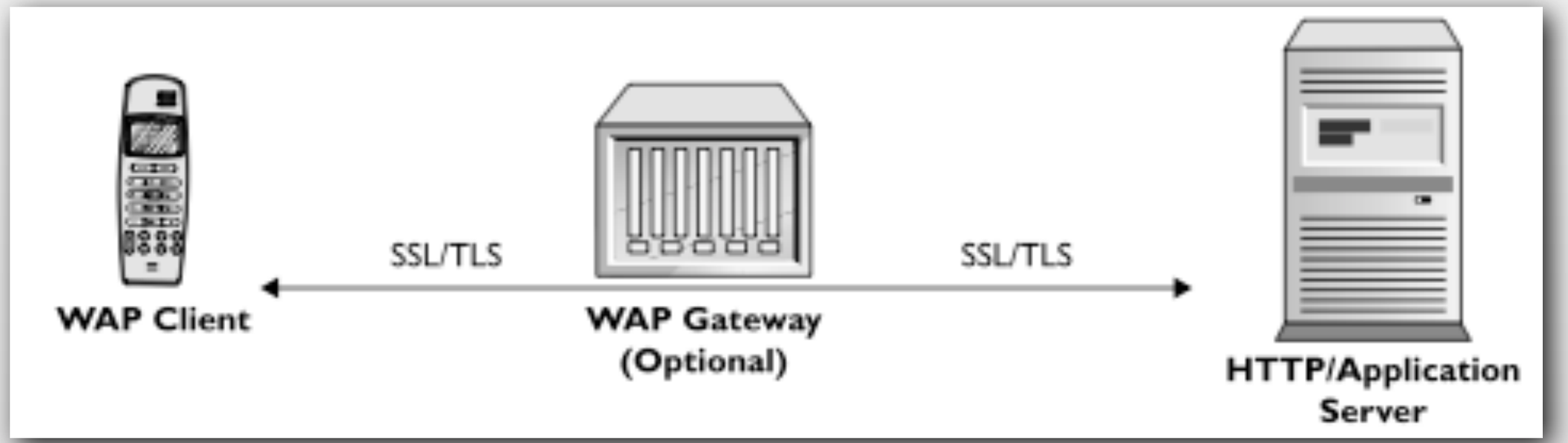
- SSL/TLS is a critical aspect of online security to keep sensitive information private over the Internet
- WAP 1.0 used TLS but not end to end due to the limited horsepower on mobile devices
- WTLS is similar to TLS; used for low-bandwidth data channels that cannot support full-blown TLS implementation



WAP 1.0 and transport encryption

Encryption

- WAP 2.0 supports full end to end TLS to eliminate the “WAP gap”
- WAP gateway optional (can be used for optimization purposes)
- WTLS is no longer needed



WAP 2.0 and transport encryption

Application Attacks on Mobile HTML Sites

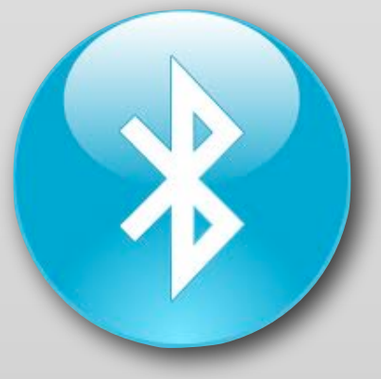
- Many traditional web application attacks will work on mobile browsers/devices supporting WAP 2.x/Mobile HTML sites
  - Cross-Site Scripting (XSS)
  - SQL Injection
  - Cross-Site Request Forgery (CSRF)
  - HTTP Redirects
  - Phishing
  - Session Fixation
  - Non-SSL Login

WAP and Mobile Browser Weaknesses

- Known limitations of WAP and mobile browsers:
  - Lack of HTTPOnly Flag Support
  - Lack of SECURE Flag Support
  - Handling Browser Cache
  - WAP Limitations

Bluetooth Security

- Overview of the Technology
- Bluetooth Security Features
  - Pairing
  - Authentication
  - Authorization
  - Confidentiality
- Threats to Bluetooth Devices and Networks
- Bluetooth Vulnerabilities
- Recommendations



Overview of the Technology

- Conceived at Ericsson Mobile Communications to create a wireless keyboard system and then adapted for more generic purposes
- Common Uses include:
  - Wireless keyboard, mouse, and printer connectivity
  - Device synchronization (phone to desktop)
  - File transfer (phone to desktop or photo printer)
  - Gaming console integration (Nintendo Wii remotes and Sony PS3 headsets)
  - Tethering for Internet access (using data-enabled mobile phone as a modem for Internet access from a laptop with Bluetooth providing inter-device connectivity)
  - Hands-free and voice-activated mobile phone kits for cars

Bluetooth Security Features - Pairing

- Pairing, the process whereby two Bluetooth devices establish a link and agree to communicate, is critical to the overall security architecture and is tightly integrated with other security features
- During pairing, the communicating devices agree on and generate keys used to identify and relate to other devices; these keys are also used for device authentication and communication encryption
- Prior to Bluetooth v2.1, pairing between devices is accomplished through the entry of a PIN with a maximum length of 128 bits.
- Bluetooth v2.1 introduced Secure Simple Pairing to improve security through the use of Elliptic Curve Diffie-Hellman for key exchange and link key generation

Bluetooth Security Features - Authentication

- Authentication is the process whereby one device verifies the identity of another device
- A traditional challenge-response mechanism is used between the *claimant* device and the *verifier* device
- Response to challenge based on a function involving a random number, the claimant's Bluetooth device address, and a secret key generated during device pairing
- To prevent repeated attacks in a limited timeframe, on an authentication failure the verifier will delay its next attempt to authenticate the claimant

Bluetooth Security Features - Authorization

- Authorization allows for decision making about resource access and connection configuration based on permissions granted a given device or service
- **Device Trust Levels:** Bluetooth devices can be “trusted” (previously been paired and have full access) or “untrusted” (not previously paired and have restricted access) in relation to other Bluetooth devices
- **Service Security Levels:**
  - Level 1 services require authentication and authorization
  - Level 2 services require authentication only
  - Level 3 services have no security and are open to all devices

Bluetooth Security Features - Confidentiality

- Confidentiality is provided through the use of encryption
- Bluetooth uses E_0, a stream cipher, as the basis for encryption and provides three different encryption modes
  - Mode 1 does not do any encryption; all traffic is unencrypted
  - Mode 2 encrypts traffic between individual endpoints but broadcast traffic is unencrypted
  - Mode 3 encrypts both broadcast and point-to-point traffic

Threats to Bluetooth Devices and Networks

- Bluetooth devices and networks are also subject to threats like eavesdropping, impersonation, denial of service, and man-in-the-middle attacks
- Additional Bluetooth threats include:
  - Location tracking
  - Key management issues
  - Bluejacking
  - Implementation issues (Bluesnarfing, Bluebugging, Car whispering)

Bluetooth Vulnerabilities

- Bluetooth Versions Prior to v1.2
  - The unit key is reusable and becomes public when used
- Bluetooth Versions Prior to v2.1
  - Short PINs are permitted
  - The encryption keystream repeats
- All Versions
  - Unknown RNG strength for challenge-response
  - Negotiable encryption key length (as small as one byte!)
  - Shared master key
  - Weak E_0 stream cipher (theoretical known-plaintext attack)

Recommendations

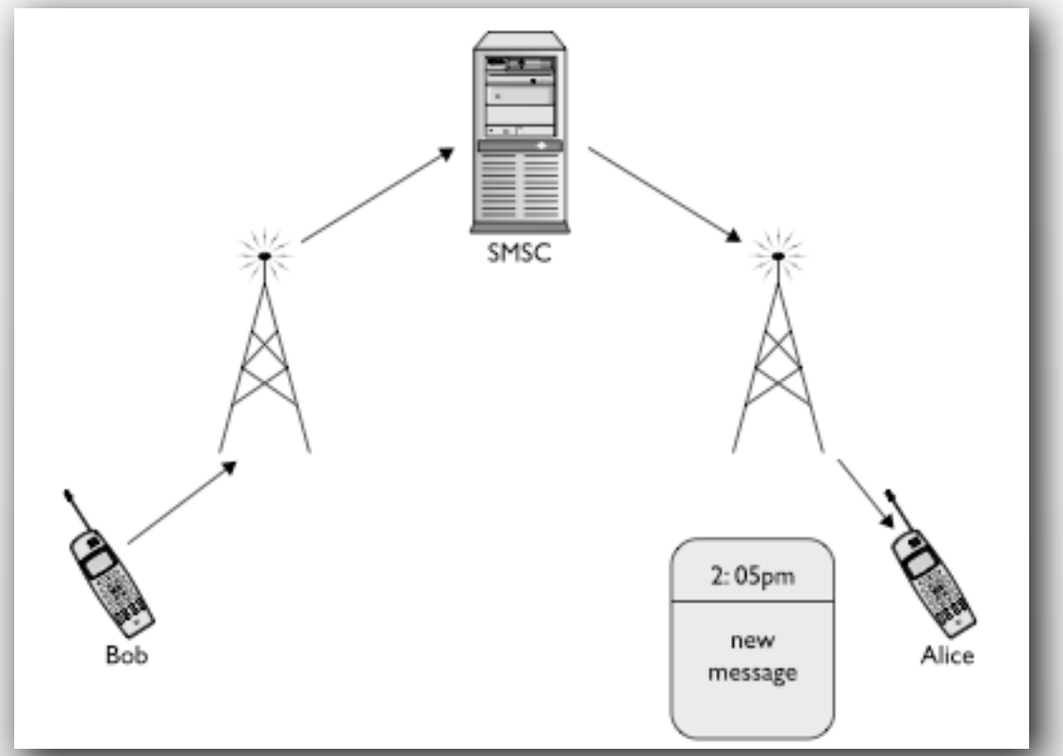
- Use complex PINs for Bluetooth devices
- In sensitive and high-security environments, configure Bluetooth devices to limit the power used by the Bluetooth radio
- Limit the services and profiles available on Bluetooth devices to only those required
- Configure Bluetooth devices as non-discoverable except during pairing
- Enable mutual authentication for all Bluetooth communications
- Configure the maximum allowable size for encryption keys
- Unpair devices that had previously paired with a device if a Bluetooth device is lost or stolen

SMS Security

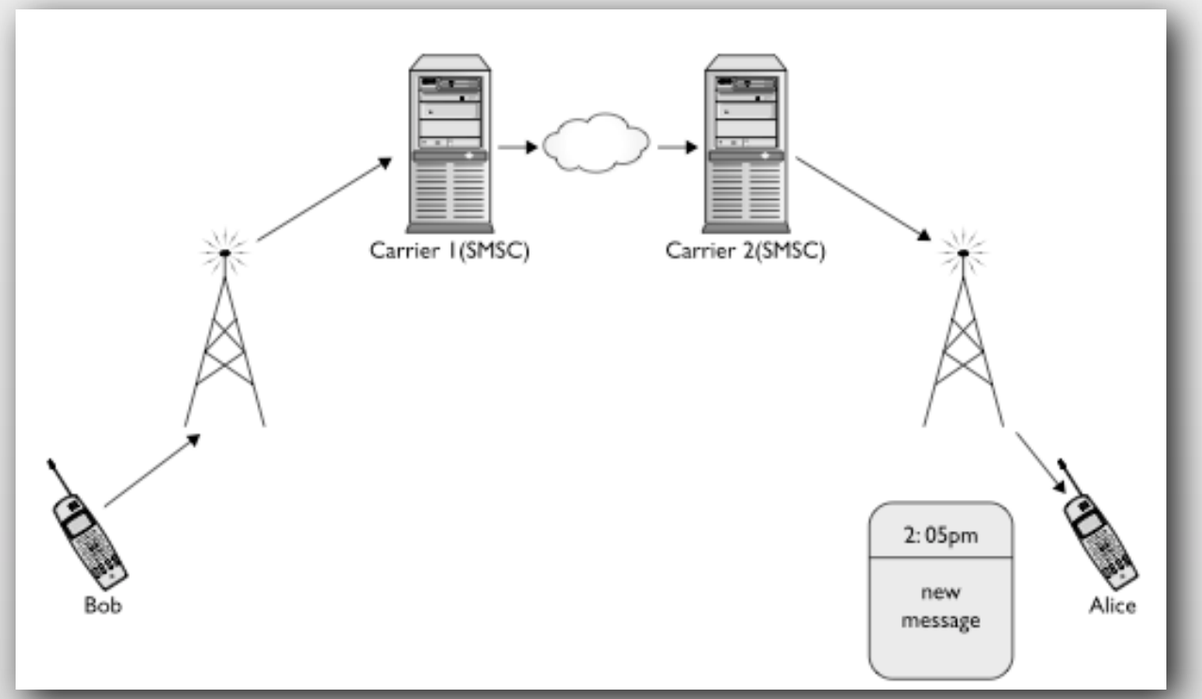
- Overview of Short Message Service
- Overview of Multimedia Messaging Service
- Protocol Attacks
- Application Attacks

Overview of Short Message Service

- SMS is designed for one mobile subscriber to send a short message (up to 160 characters) to another mobile subscriber



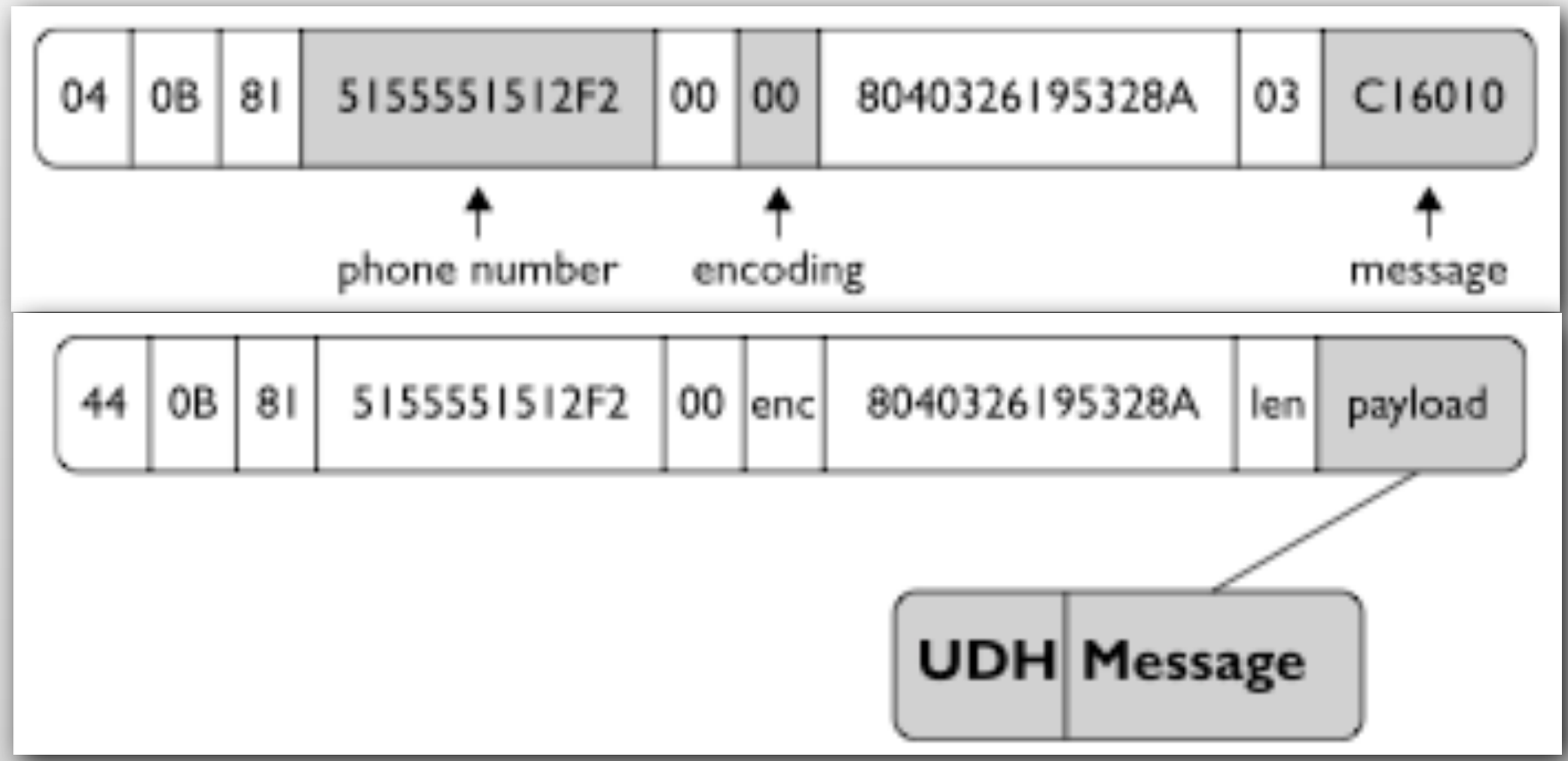
SMS message between phones using the same carrier



SMS message between phones on different carriers

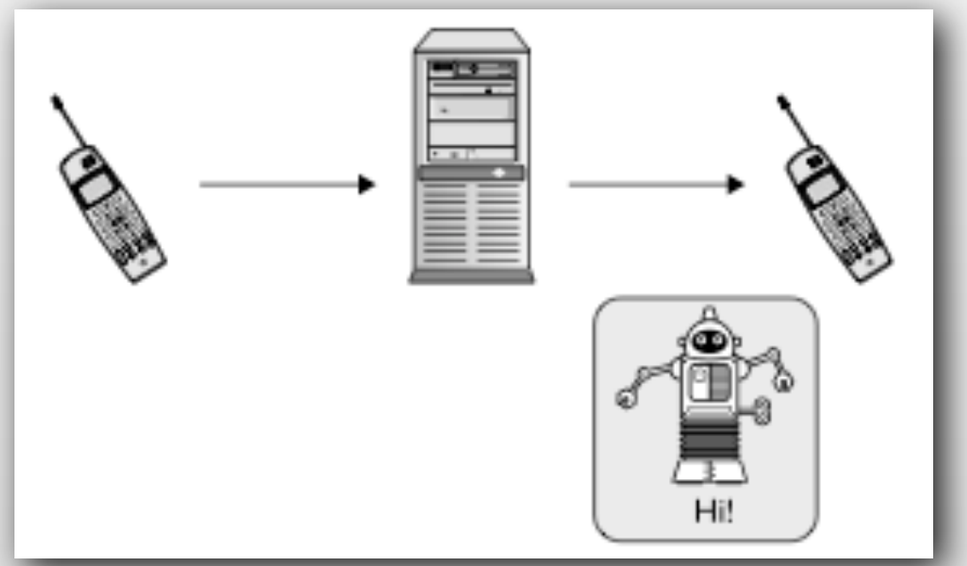
Overview of Short Message Service

- A raw SMS message is known as a *Protocol Data Unit* (PDU)
- A basic SMS PDU contains several header fields as wells as message contents

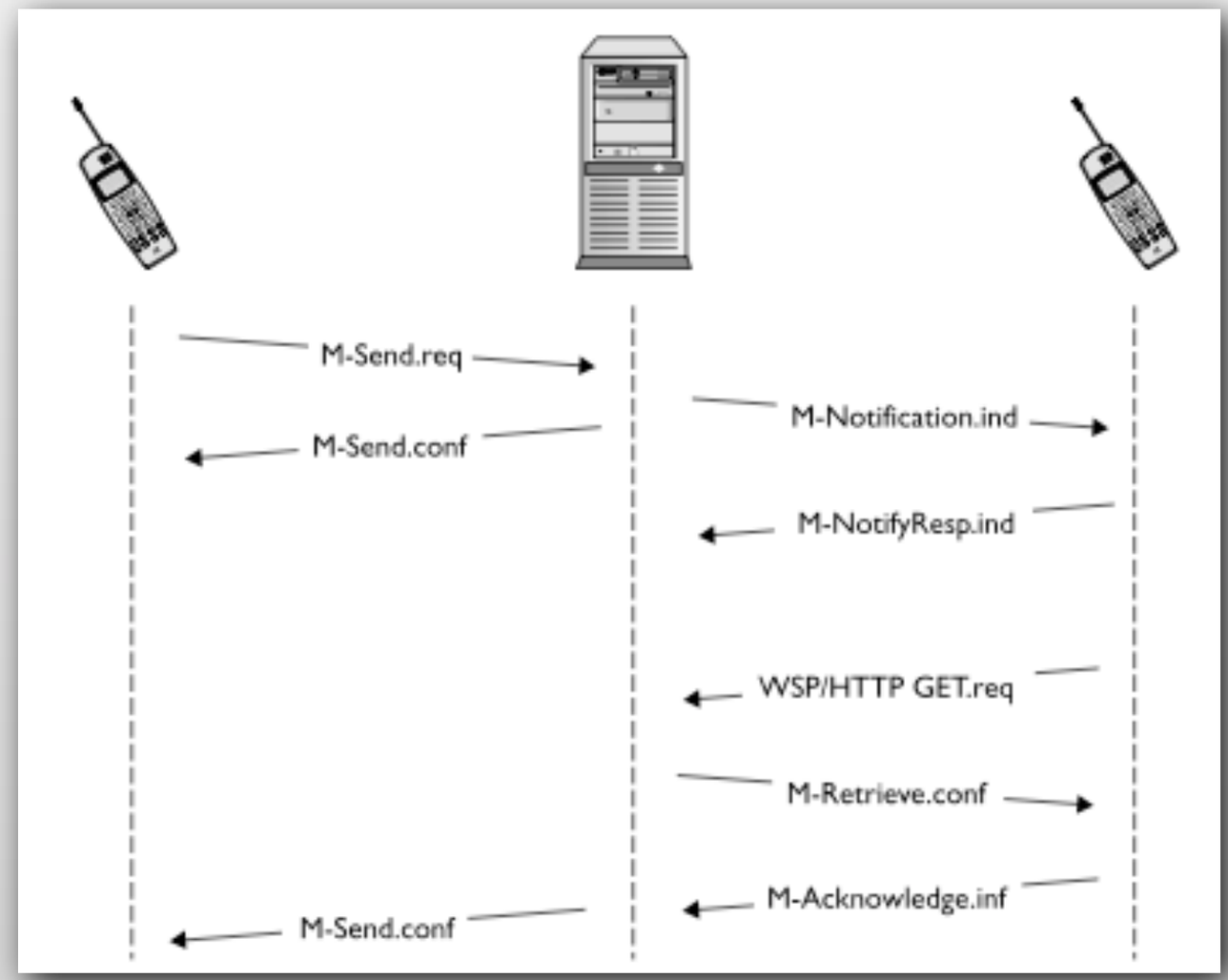


Overview of Multimedia Messaging Service

- MMS can send various types of images, audio, and video in addition to text
- MMS is fundamentally different from SMS although they may look exactly the same from a user's perspective



MMS from a user standpoint



Detailed MMS diagram

Protocol Attacks

- Abusing Legitimate Functionality
    - Attacks targeting functionality that is meant to be hidden from the end user. Ex: Administrative and provisioning communications such as updates and voicemail notifications.
- Attacking Protocol Implementations
    - Attacks targeting vulnerabilities in the implementations of the popular SMS protocols with the intent of sending a corrupted message to a victim's phone resulting in the phone running hostile code.

Protocol Attacks - Abusing Legitimate Functionality

- WAP Push Attack
- MMS Notification
- Battery-Draining Attack
- Silent Billing Attack
- OTA Settings Attack

Application Attacks

- Targets applications that use SMS as a delivery mechanism
- Unlike protocol attacks that are mostly version agnostic, application attacks are very specific to software versions running on phones
- Application vulnerabilities tend to fall into browser, MMS client, or image categories
- Examples:
  - iPhone Safari vulnerability results in heap overflow after viewing a malicious page within mobile Safari, allowing attacker to execute arbitrary code on the iPhone
  - Motorola RAZR JPG overflow vulnerability due to the way the RAZR parsed thumbprints in the JPG EXIF header, allowing arbitrary code execution using malicious JPG image

Mobile Geolocation

- Geolocation Methods
- Geolocation Implementation
  - Android
  - iPhone
- Risks of Geolocation Services
  - End User
  - Service Providers
- Geolocation Best Practices

Geolocation Methods

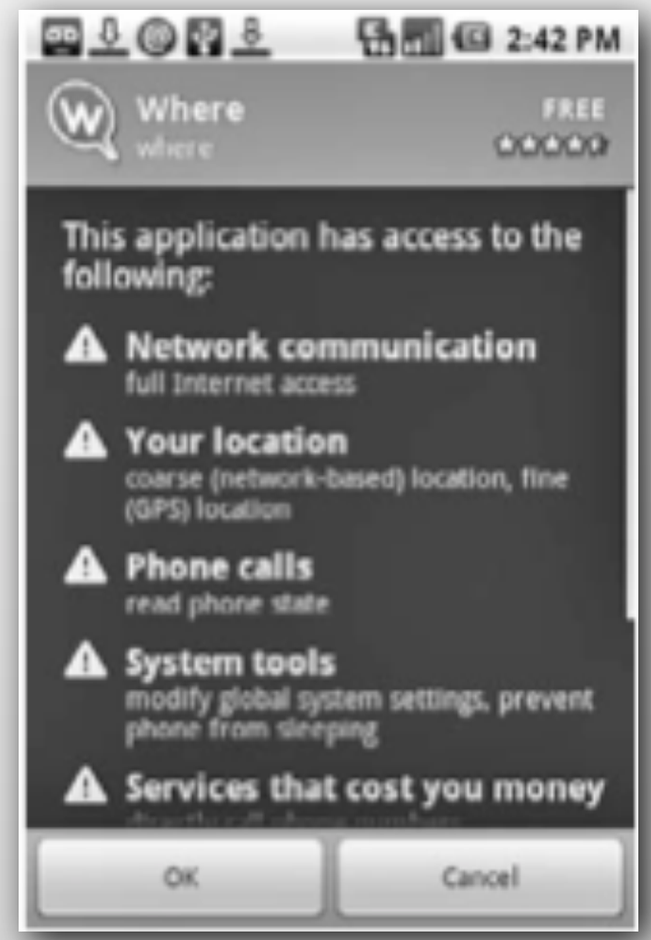
- Tower Triangulation (Accuracy: 50m - 1,000m)
  - Oldest widely used method of geolocation via cell phone
  - Uses relative power levels of radio signals between cell phone and cell tower; requires at least two cell towers
  - Fairly inexact due to distance from cell towers and signal strength
- GPS (Accuracy: 5m - 15m)
  - Uses satellite signals instead of cell phone or wireless infrastructure; reception may be poor indoors
  - Can provide continuous tracking updates, useful for real-time applications

Geolocation Methods

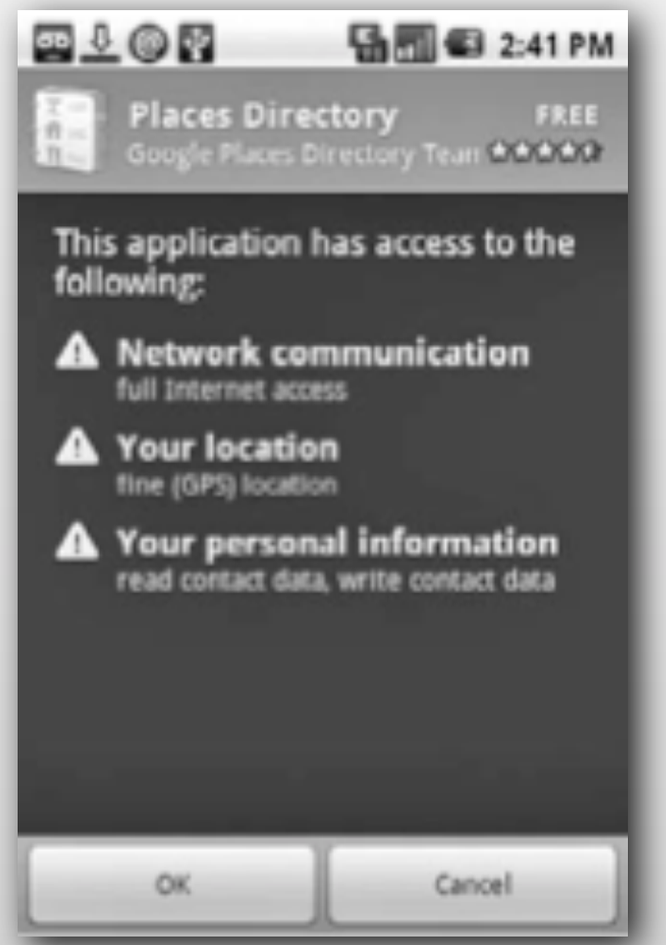
- 802.11 (Accuracy: 10m - 200m but potentially erroneous)
  - iPhone was the first smartphone to use this approach, using an API from Skyhook Wireless which uses data from wireless access points to create a large “wardriving” database
  - Allows for devices without GPS to get potentially highly accurate location data
  - Faster and more accurate than tower triangulation
  - Drawbacks due to dependency on wireless access points which could be moved
  - Google’s “Latitude” service provides a newer implementation of Skyhook’s technology

Geolocation Implementation - Android

- Permission to use geolocation features is requested via the program manifest and is granted by the user at install time
- ACCESS_COARSE_LOCATION (for cell triangulation or Wi-Fi)
- ACCESS_FINE_LOCATION (for GPS)



A permissions request for coarse and fine location services



A permissions request for only fine location services

Geolocation Implementation - iPhone

- Requires user approval every time an application that uses geolocation APIs is launched
- CLLocationAccuracyBest
- CLLocationAccuracyNearestTenMeters
- CLLocationAccuracyHundredMeters
- CLLocationAccuracyThreeKilometers

The iPhone location permissions dialog

Risks of Geolocation Services - End User

- Positional data stored on remote servers, when it can be tied to an individual, introduces a new avenue for data theft
- Along with other sensitive data, not only could this be a breach of user privacy, but also a potential source of information in court
- Broadcasting user's location voluntarily (think Foursquare) may also lead to stalking or harassment
- Few points to ponder:
  - Privacy and data retention policies for positional information
  - Third-party sharing and data transfer channels
  - Course of action for law enforcement requests

Risks of Geolocation Services - Service Providers

- Risk of negative publicity from a data breach, legal or congressional subpoenas, and potential assistance to criminal acts by allowing third parties to track individual users
- Often times, the stored geolocation data is not really necessary to provide the required functionality
- Legal obligation to follow privacy guidelines in countries like the UK ("Data Protection Act")

Geolocation Best Practices

- Use the least precise measurement necessary
- Discard data after use
- Keep data anonymous
- Indicate when tracking is enabled
- Use an opt-in model
- Have a privacy policy
- Do not share geolocation data with other users or services
- Familiarize yourself with local laws

Enterprise Security on the Mobile OS

- Device Security Options
  - PIN
  - Remote Wipe
- Secure Local Storage
- Encryption
- Application Sandboxing
- Application Signing
- Buffer Overflow Protection

Device Security Options - PIN

- Enabling the PIN is the first step in securing a mobile device
- Unmotivated attacker could wipe and sell it instead of trying to break into the OS
- Data on the device (or data that phone has access to) is at times worth more than the device
- Although a four-digit PIN only needs 10,000 attempts to brute-force it, many mobile devices have a time delay after ten failed attempts
- On some devices like the iPhone, the SIM card also has PIN protection

Device Security Options - Remote Wipe

- The ability to remote wipe data on a mobile device (especially if its a corporate one) is imperative
- Remote wipe functionality makes the loss of such devices a lot less stressful
- Both iPhone and Android support remote wipe functionality

Secure Local Storage

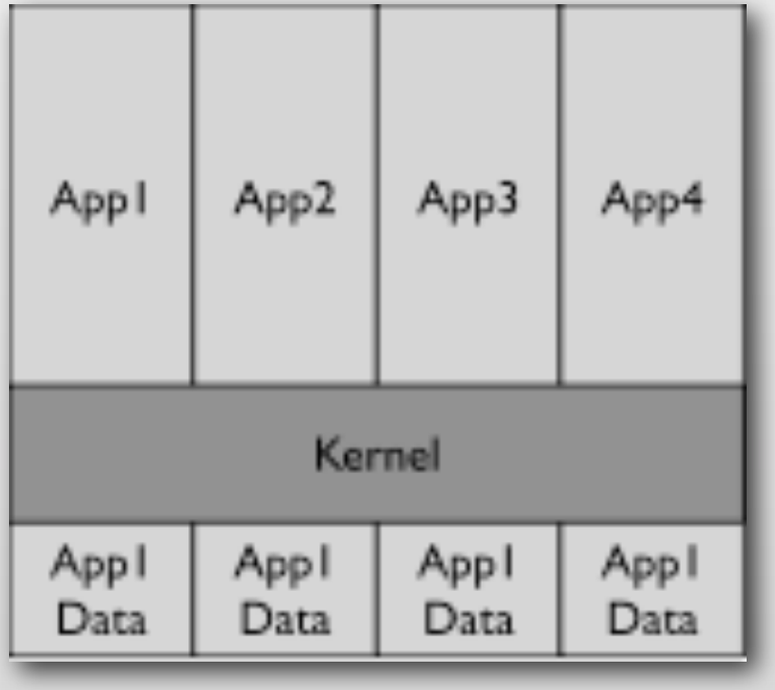
- Ability to store sensitive information locally in a secure fashion is also an imperative security feature for mobile devices
- Many applications store login information, such as username and password, locally on the device in clear text (without encryption)
- The iPhone addresses this need via the use of “Keychain” which can be used to store, retrieve, and read sensitive information, such as passwords, certificates, and secrets

Encryption

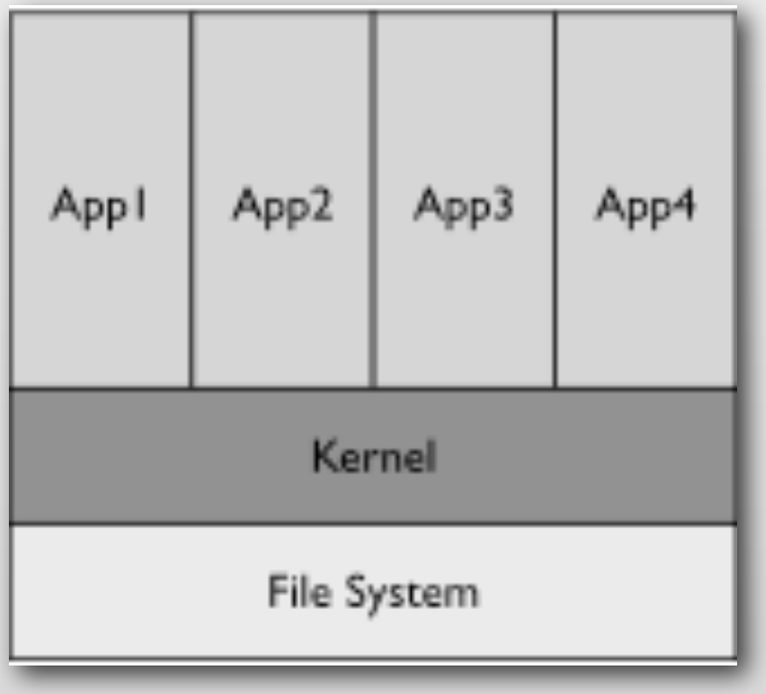
- Full Disk Encryption
  - Unlike desktop OSes, mobile OSes have little or no solutions for full disk encryption.
  - *iOS4 and Android ICS supposedly have this feature*
- Email Encryption
  - None of the most popular mobile OSes provide native support for local email
  - *Good for Enterprise supports both iPhone and Android*
- File Encryption
  - Most major mobile OSes support file encryption

Application Sandboxing

- The primary goals are:
  - ensure one application is protected from another
  - protect the underlying OS from the application
  - ensure a bad application is isolated from the good ones



New application isolation model



Traditional application isolation model

Application Signing

- It is a vetting process to provide users some level of assurance concerning the application and to associate authorship and privileges to an application
- It is *not* a measure of the security of the application or its code
- Depending on whether or not the application is signed, and what type of certificate is used, different privileges are granted on the OS
- Most mobile OSes, including the iPhone and Android, require application signing; Android allows self-signed certificates whereas the iPhone does not
- It is assumed that malware authors would not be able to bypass the appropriate levels of controls by a signing authority to get a certificate

Buffer Overflow Protection

- Major attack class for mobile OSes written in C, Objective-C, or C++
- The iPhone mitigates buffer overflows by making the stack and heap on the OS non-executable; any attempts to do so would cause application exception
  - Stack-based protection on the iPhone is performed using the NX Bit that marks certain areas of memory as non-executable
- The Android OS mitigates buffer overflow attacks by leveraging the use of ProPolice, OpenBSD malloc/calloc, and the safe_iop function
  - ProPolice provides stack smasher protection; OpenBSD's malloc/calloc make heap-based buffer overflows more difficult; safe_iop library provides functions to perform safe integer operations

But Wait...

THERE'S MORE!

OWASP Mobile Security Project

Our primary focus is at the application layer. While we take into consideration the underlying mobile platform and carrier inherent risks when threat modeling and building controls, we are targeting the areas that the average developer can make a difference. Additionally, we focus not only on the mobile applications deployed to end user devices, but also on the broader server-side infrastructure which the mobile apps communicate with. We focus heavily on the integration between the mobile application, remote authentication services, and cloud platform-specific features.

Top 10 Mobile Risks

1. Insecure Data Storage
2. Weak Server Side Controls
3. Insufficient Transport Layer Protection
4. Client Side Injection
5. Poor Authorization and Authentication
6. Improper Session Handling
7. Security Decisions Via Untrusted Inputs
8. Side Channel Data Leakage
9. Broken Cryptography
10. Sensitive Information Disclosure

1. Insecure Data Storage

- Store ONLY what is absolutely required
- Never use public storage areas (like SD card)
- Leverage secure containers and platform-provided encryption APIs
- Do not grant files world-readable or world-writeable permissions

2. Weak Server Side Controls

- Understand the additional risks mobile apps introduce into existing architectures
- Leverage the wealth of knowledge that is already out there
- OWASP Web Top 10, Cloud Top 10, Web Services Top 10
- Cheat sheets, development guides, ESAPI

3. Insufficient Transport Layer Protection

- Ensure that all sensitive data leaving the device is encrypted
- This includes data over carrier networks, WiFi, and even NFC
- When security exceptions are thrown, it's generally for a reason -- DO NOT ignore them!

4. Client Side Injection

- Sanitize or escape untrusted data before rendering/executing it
- Use prepared statements for database calls; concatenation is still bad, and always will be bad
- Minimize the sensitive native capabilities tied to hybrid web functionality

5. Poor Authorization and Authentication

- Contextual info can enhance things, but only as part of multi-factor implementation
- Out-of-band doesn't work when it's all the same device
- Never use device ID or subscriber ID as sole authenticator

6. Improper Session Handling

- Don’t be afraid to make users re-authenticate every so often
- Ensure that tokens can be revoked quickly in the event of a lost/ stolen device
- Utilize high entropy, tested token generation resources

7. Security Decisions Via Untrusted Inputs

- Check caller's permissions at input boundaries
- Prompt the user for additional authorization before allowing
- Where permission checks cannot be performed, ensure additional steps required to launch sensitive actions

8. Side Channel Data Leakage

- Never log credentials, PII, or other sensitive data to system logs
- Remove sensitive data before screenshots are taken, disable keystroke logging per field, and utilize anti-caching directives for web content
- Debug your apps before releasing them to observe files created, written to, or modified in any way
- Carefully review any third-party libraries you introduce and the data they consume
- Test your applications across as many platform versions as possible

9. Broken Cryptography

- Storing the key with the encrypted data negates everything
- Leverage battle-tested crypto libraries instead of writing your own
- Take advantage of what your platform already provides!

10. Sensitive Information Disclosure

- Private API keys are called that for a reason -- keep them away from the client
- Keep proprietary and sensitive business logic on the server
- Almost never a legitimate reason to hardcode a password