

# Ch 7: Mobile Device Management



## CNIT 128: Hacking Mobile Devices

Updated 4-4-17

# What is MDM?

- Frameworks that control, monitor, and manage mobile devices
- Deployed across enterprises or service providers
- Provide these functions remotely:
  - Monitoring
  - Control
  - Manage

# MDM Frameworks

- Policies and features that administrators can control and enforce
- iOS, Android and BlackBerry
  - Each provide their own MDM frameworks
- MDM solutions from third parties
  - MobileIron
  - AirWatch
  - BlackBerry Enterprise

# MDM Frameworks

- Policies and features that administrators can control and enforce
- iOS, Android and BlackBerry
  - Each provide their own MDM frameworks
  - Third-party vendors develop products that use the frameworks

# Top Five MDM Solutions (2015)

1. MobileIron
  2. AirWatch
  3. Citrix (XenMobile)
  4. IBM (MaaS360)
  5. Good Technology
    - Provides MDM without leveraging platform framework and support
- [Link Ch 7a](#)

# Project: MaaS360 from IBM

← → ↻ 🏠 <https://m3.maas360.com/emc/>

**MaaS360**  
by IBM

## Quick Start

1. Select Platforms (Setup pending for iOS) ✓

2. Add Devices

### Add a Device

A message will be sent to the user with a one-time passcode to add a device.

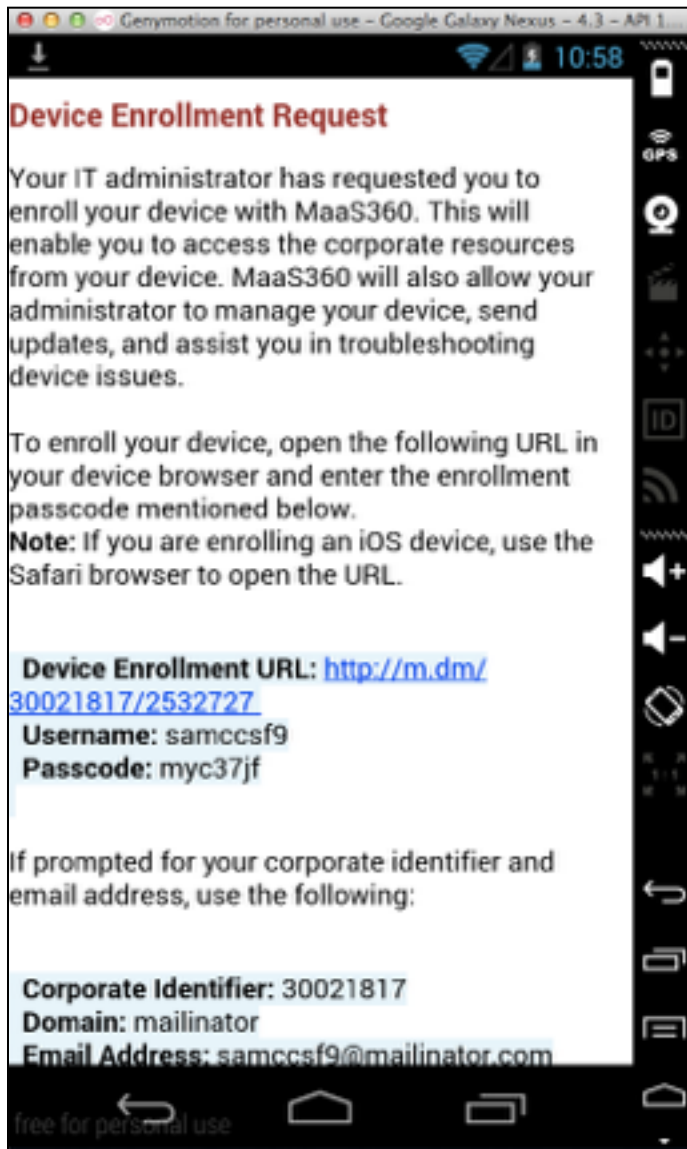
Username -  \*

Email Address -

Domain -

Phone Number

# Project: MaaS360 from IBM



Genymotion for personal use - Google Galaxy Nexus - 4.3 - API 1

10:58

## Device Enrollment Request

Your IT administrator has requested you to enroll your device with MaaS360. This will enable you to access the corporate resources from your device. MaaS360 will also allow your administrator to manage your device, send updates, and assist you in troubleshooting device issues.

To enroll your device, open the following URL in your device browser and enter the enrollment passcode mentioned below.

**Note:** If you are enrolling an iOS device, use the Safari browser to open the URL.

**Device Enrollment URL:** <http://m.dm/30021817/2532727>

**Username:** samccsf9  
**Passcode:** myc37jf

If prompted for your corporate identifier and email address, use the following:

**Corporate Identifier:** 30021817  
**Domain:** mailinator  
**Email Address:** samccsf9@mailinator.com

free for personal use



Genymotion for personal use - Google Galaxy Nexus - 4.3 - API 1

10:59

## MaaS360<sup>®</sup> by IBM

MaaS360 MDM for Android  
MaaS360

13.35MB/21.86MB 61%

### Users also installed

|  |  |  |
|--|--|--|
| <br>MaaS360 MDM for<br>***** FREE | <br>MaaS360 Browser<br>***** FREE | <br>IBM Mobile Client<br>***** FREE |
|--|--|--|

free for personal use

# Project: MaaS360 from IBM

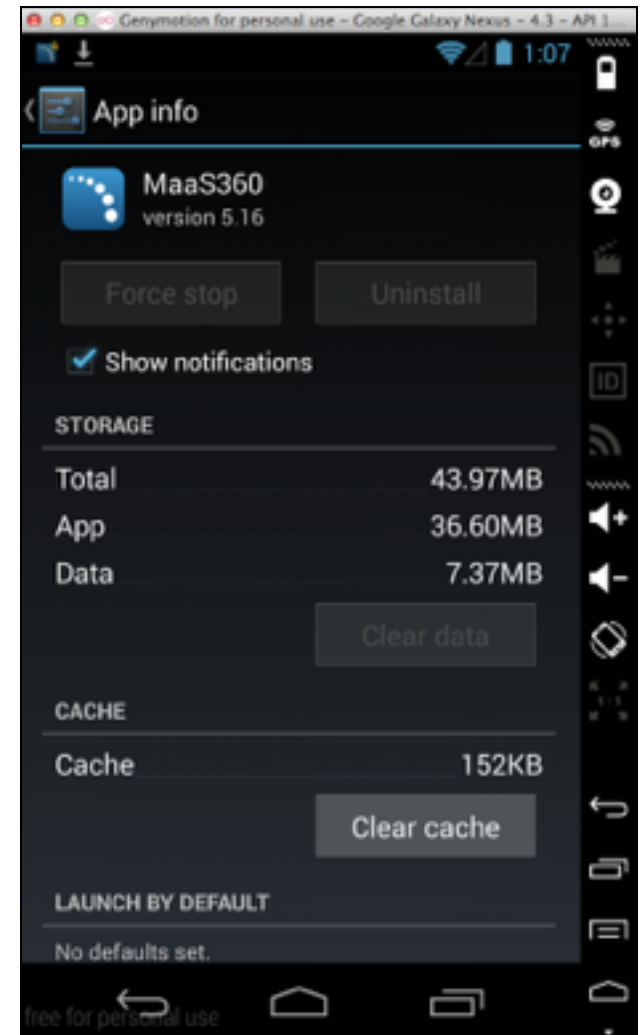
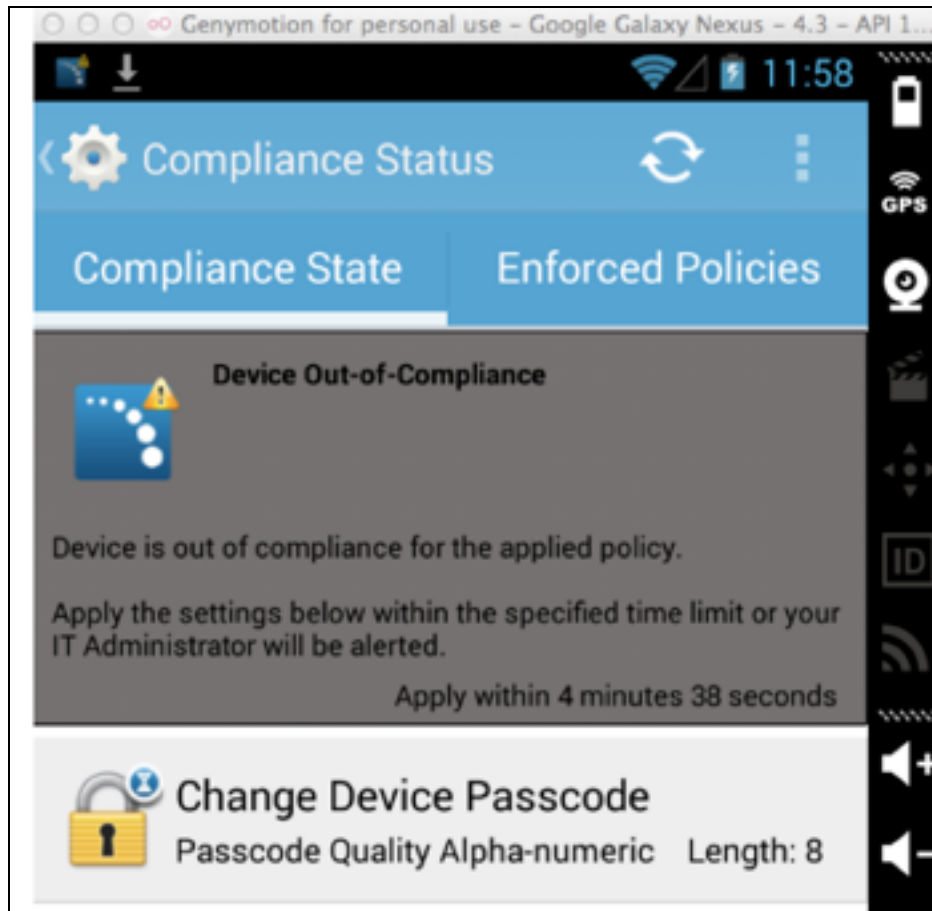
The screenshot displays the MaaS360 management interface. At the top, the navigation menu includes DEVICES, USERS, SECURITY (highlighted), APPS, DOCS, REPORTS, and SETUP. The current page is titled "Alpha Passcode" and shows a "Last Published: [Version:N.A.]" status. The left sidebar lists various settings: Device Settings, Passcode (selected), Security, Restrictions, Application Compliance, Native App Compliance, ActiveSync, and Wi-Fi. The main content area is titled "Configure Passcode Policy" and includes a "Passcode Settings" section with the following configurations:

| Setting   | Value                               | OS Support   |
|---|-------------------------------------|--------------|
| Configure Passcode Policy   | <input checked="" type="checkbox"/> | Android 2.2+ |
| Passcode Quality  | Alphanumeric                        | Android 2.2+ |
| Minimum Passcode Length (4-16 characters)                           | 8                                   | Android 2.2+ |
| Maximum Passcode Age (In Days)                                      |                                     | Android 3.0+ |
| Allowed Idle Time (In minutes) Before Auto-Lock                     | 30 minutes                          | Android 2.2+ |
| Passcode history  |                                     | Android 3.0+ |
| Number of Failed Passcode Attempts Before All Data is Erased (4-16) | 10                                  | Android 2.2+ |

At the bottom of the page, there is a "Return to Quick Start" button, a user login area with "Username: samccs9@mailinator.com | Account ID: 30021817", and a "PRIVACY AND LEGAL" link.



# Project: MaaS360 from IBM



# Project : MaaS360 from IBM

The screenshot displays the MaaS360 web interface for managing a mobile device. The browser address bar shows the URL <https://m3.maas360.com/emc/#>. The navigation menu includes DEVICES, USERS, SECURITY, APPS, DOCS, REPORTS, and SETUP. The device being managed is identified as 'samccsf3-Google Galaxy Nexus - 4.3 - API 18 - 720x1280'. A 'More' dropdown menu is open, listing actions such as Lock, Reset Passcode, Selective Wipe, Wipe, Change Policy, Change Rule Set, Distribute App, Distribute Doc, Remove Control, Hide, and Request Data Refresh.

| Summary      |                  | Hardware Inventory |                     |
|--------------|------------------|--------------------|---------------------|
| Username     | samccsf3         | Operating System   | Android 4.3 (JLS)   |
| Manufacturer | Genymotion       | Model              | Google Galaxy Nexus |
| IMEI/MEID    | 0000000000000000 | Device ID          | androidc259148      |
| Ownership    | Not Defined      |                    |                     |

| WorkPlace & Security    |                        |                         |  |
|-------------------------|------------------------|-------------------------|--|
| Managed Status          | Enrolled ✓             | Applied Policy          | MDM: Alpha pas                                   |
| Last Reported           | 03/18/2015 16:04 EDT ✓ | Jailbroken/Rooted       | Yes 1  |
| Failed Settings         | No ✓                   | Selective Wipe Status   | Not Applied ✓                                    |
| Encryption Level        | No Encryption 1        | Passcode Status         | MDM: Not Compliant 1<br>WorkPlace: Not Enabled 1 |
| Policy Compliance State | Out of Compliance 1    | Rules Compliance Status | -  |
| Rule Set Name           | -                      |                         |  |

Return to Quick Start | Username: samccsf3@mailinator.com | Account ID: 30021828 | PRIVACY AND LEGAL

# Categories of MDM Frameworks

# Device-Centric Model

- Leverages platform capabilities and feature sets
- To configure, secure, and harden the mobile device
- Uses underlying framework to detect changes in device security and configuration
- Includes: MobileIron, AirWatch, and Tangoe

# Data-Centric Model

- Focuses on securing data/content of interest
  - Not on controlling or securing the whole device
- Ensures security and integrity of data
  - Without relying on platform capabilities
- Relies on a custom mobile app
- Example: Good for Enterprise

# Hybrid Model

- Combines platform MDM framework with solution-specific features
- Protects data and provides device management

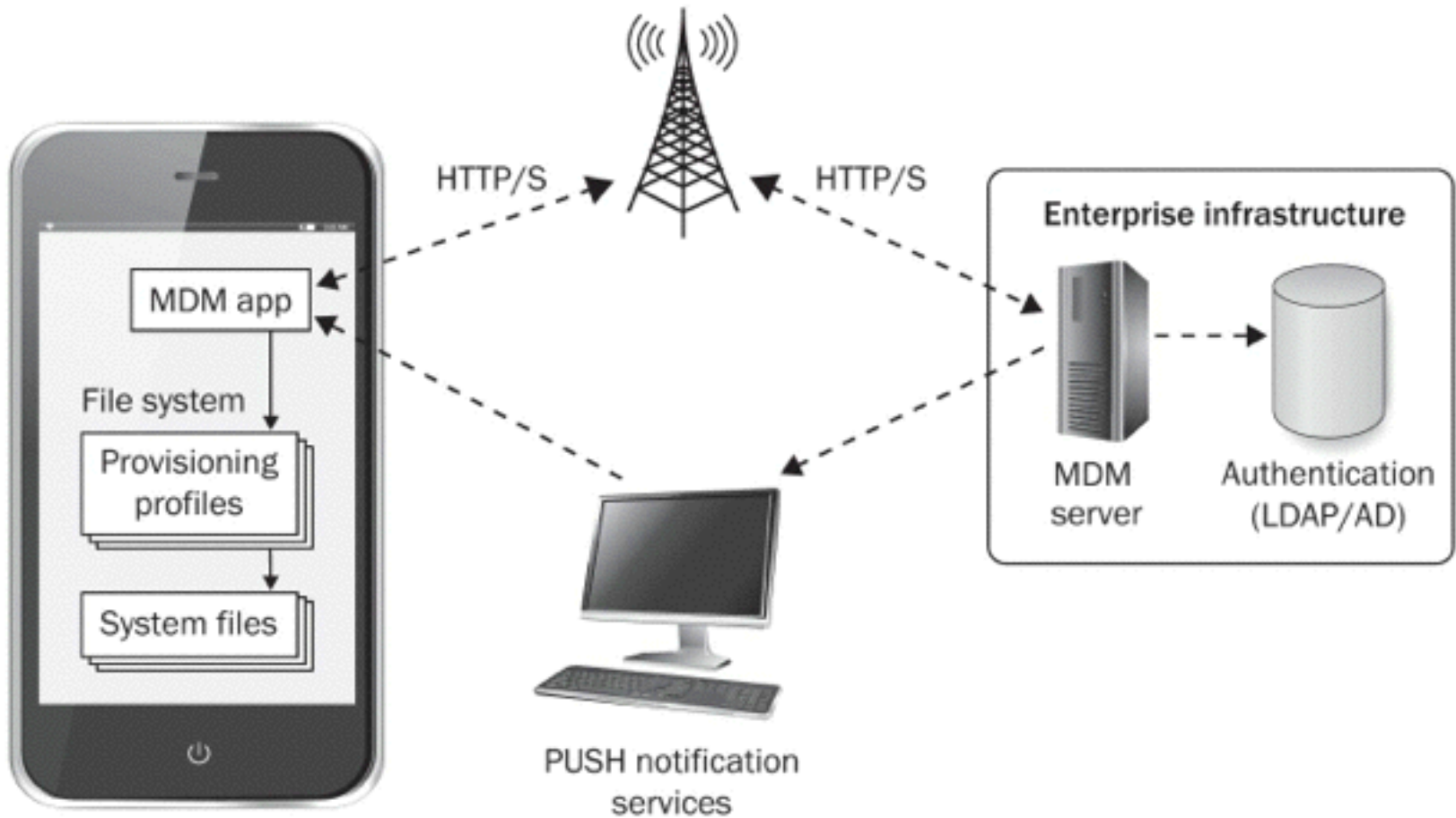
# Common Features

# Device Provisioning

- Deploy and enforce policies and restrictions on mobile devices
- Provide access to resources controlled by the MDM server
- Often use MDM client apps



# Device Provisioning



# Provisioning Profile

- Installed on the device by the MDM client
- An XML or text file
- Specifies configuration and provisioning information for the mobile device
- May be
  - Plan-text
  - Signed
  - Encrypted & signed

# Policy Enforcement

1. Device receives a provisioning profile; it's verified and decrypted, then parsed
2. Mobile platform populates system files with the configuration information required to enforce policies from the provisioning profile
3. System files are parsed by system services and implement the configuration settings

# iOS

- MDM server sends provisioning profiles through Apple's Mail client (ActiveSync) or the MDM app installed on the device
- Mobile device stores the profiles at
  - `/private/var/mobile/Library/ConfigurationProfiles`
- Stored as XML files (plist) with **.stub** file extensions

# Example Provisioning Profile

- Plist files with .stub extensions

```
iPhone:/private/var/mobile/Library/ConfigurationProfiles root# ls -l *.stub
-rw-r--r-- 1 mobile mobile  2516 Apr 17  2012
4598b7ba178f96bae7864be9b88a1545bc3296eaa+800194199.stub
-rw-r--r-- 1 mobile mobile  7533 Oct 17  2011 com_apple_attwifi+3369864630.stub
-rw-r--r-- 1 mobile mobile 35057 Jan  6 10:36
com_good_iphone_policy+1281327003.stub
-rw-r--r-- 1 mobile mobile  2962 Dec  8  2011
f9ba36a2a2360ede0d588fe242bfdbc7cd12c169a+28338739.stub
```

# Sample of iOS Provisioning Profile

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  .
  .
  <dict>
    <key>MCProfileIsRemovalStub</key>
    <true/>
    <key>PayloadContent</key>
    <dict>
      <key>ConfirmInstallation</key>
      <false/>
      <key>DeviceAttributes</key>
      <array>
        <string>UDID</string>
        <string>IMEI</string>
        <string>ICCID</string>
        <string>VERSION</string>
        <string>PRODUCT</string>
      </array>
      <key>EnrollmentIdentityPersistentID</key>
      <data>
        aWRudXXXXXXXXXXXXXg
      </data>
      <key>URL</key>
      <string>https://www.xyz.com/abc.do</string>
    </dict>
  </dict>
</plist>
```

# Sample of iOS Provisioning Profile

```
    </dict>
    <key>PayloadDescription</key>
    <string>Install to enroll to encrypted profile service.</string>
    <key>PayloadDisplayName</key>
    <string>iPhone - Security Profile</string>
    <key>PayloadType</key>
    <string>Profile Service</string>
    <key>PayloadUUID</key>
    <string>xxxxx-xxx-xxxx-xxxx-xxxxxxxx</string>
    <key>ProductVersion</key>
    <string>5.1.1</string>
    <key>ProfileData</key>

    <key>ProfileTrustLevel</key>
      <integer>2</integer>
    <key>ProfileWasEncrypted</key>
    <false/>
    <key>ProfileWasSigned</key>
    <true/>
    <key>ProfileWasTrusted</key>
    <true/>
    <key>SignerCerts</key>
```

# Provisioning Process

- Verify and store provisioning profile in .stub files
- Profile installation
  - Parsing the profile
  - Updating appropriate system files to enforce policies



# System Files Populated in iOS

```
iPhone:/private/var/mobile/Library/ConfigurationProfiles/PublicInfo root# ls -l
-rw-r--r-- 1 mobile mobile 5206 Jan  6 11:36 EffectiveUserSettings.plist
-rw-r--r-- 1 mobile mobile  243 Sep 12 16:04 MCMeta.plist
-rw-r--r-- 1 mobile mobile 5970 Jan  6 11:13 Truth.plist
SG:/private/var/mobile/Library/ConfigurationProfiles root# ls -l
-rw-r--r-- 1 mobile mobile 8032 Jan  6 14:43 ProfileTruth.plist
```

- EffectiveUserSettings.plist and
- Truth.plist
  - System files that determine an iOS device's security posture
  - Truth.plist contains PIN/passcode policies, device restrictions, device timeout, etc.

# Truth.plist

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>assignedObject</key>
    <dict/>
    .
    .
    .
    <key>forcePIN</key>
    <dict>
        <key>preference</key>
        <true/>
        <key>value</key>
        <false/>
    </dict>
    <key>requireAlphanumeric</key>
    <dict>
        <key>preference</key>
        <false/>
    </dict>
</dict>
```

# Truth.plist

```
<key>restrictedValue</key>
<dict>
  <key>maxFailedAttempts</key>
  <dict>
    <key>preferSmallerValues</key>
    <true/>
    <key>value</key>
    <integer>11</integer>
  </dict>
  <key>maxGracePeriod</key>
  <dict>
    <key>preferSmallerValues</key>
    <true/>
    <key>value</key>
    <integer>3000</integer>
  </dict>
  <key>maxInactivity</key>
  <dict>
    <key>preferSmallerValues</key>
    <true/>
    <key>value</key>
    <integer>3000</integer>
  </dict>
</dict>
```

# Provisioning Status

- Once the system files repopulated
- Profile is considered installed
- Status updated to the MDM server
- Server grants access to protected resources

# Android

- MDM for Android works the same way
- Files are different in details

# Bypassing MDM

# Modifying MDM Policy Files

- On a jailbroken or rooted device
- Any user with sudo or root permission can modify system files
- Can modify **Truth.Plist**
  - To change passcode restrictions
- **Mitigation**
  - Proprietary jailbreak detection capabilities

# Enabling Simple Numerical Passcode

- Set `allowSimple` to `true`
- Set `requireAlphaNumeric` to `false`

```
<key>allowSimple</key>
  <dict>
    <key>preference</key>
    <true/>
  </dict>

  <key>requireAlphaNumeric</key>
  <dict>
    <key>preference</key>
    <false/>
  </dict>
```



# Enabling No Passcode

- Set **forcePIN** to **false**

```
<key>forcePIN</key>  
  <dict>  
    <key>preference</key>  
    <true/>  
    <key>value</key>  
    <false/>  
  </dict>
```

# Increase Number of Failed PIN Attempts

- Set `maxFailedAttempts` to a higher value

```
<key>maxFailedAttempts</key>  
<dict>  
<key>preferSmallerValues</key>  
<true/>  
<key>value</key>  
<integer>11</integer>  
</dict>
```

# Set Inactivity and Lock Grace Period

- If a device locks due to inactivity, it can be unlocked without a passcode if it's within the **grace period**

```
<key>maxGracePeriod</key>
<dict>
  <key>preferSmallerValues</key>
  <true/>
  <key>value</key>
  <integer>3000</integer>
</dict>
<key>maxInactivity</key>
<dict>
  <key>preferSmallerValues</key>
  <true/>
  <key>value</key>
  <integer>3000</integer>
</dict>
```

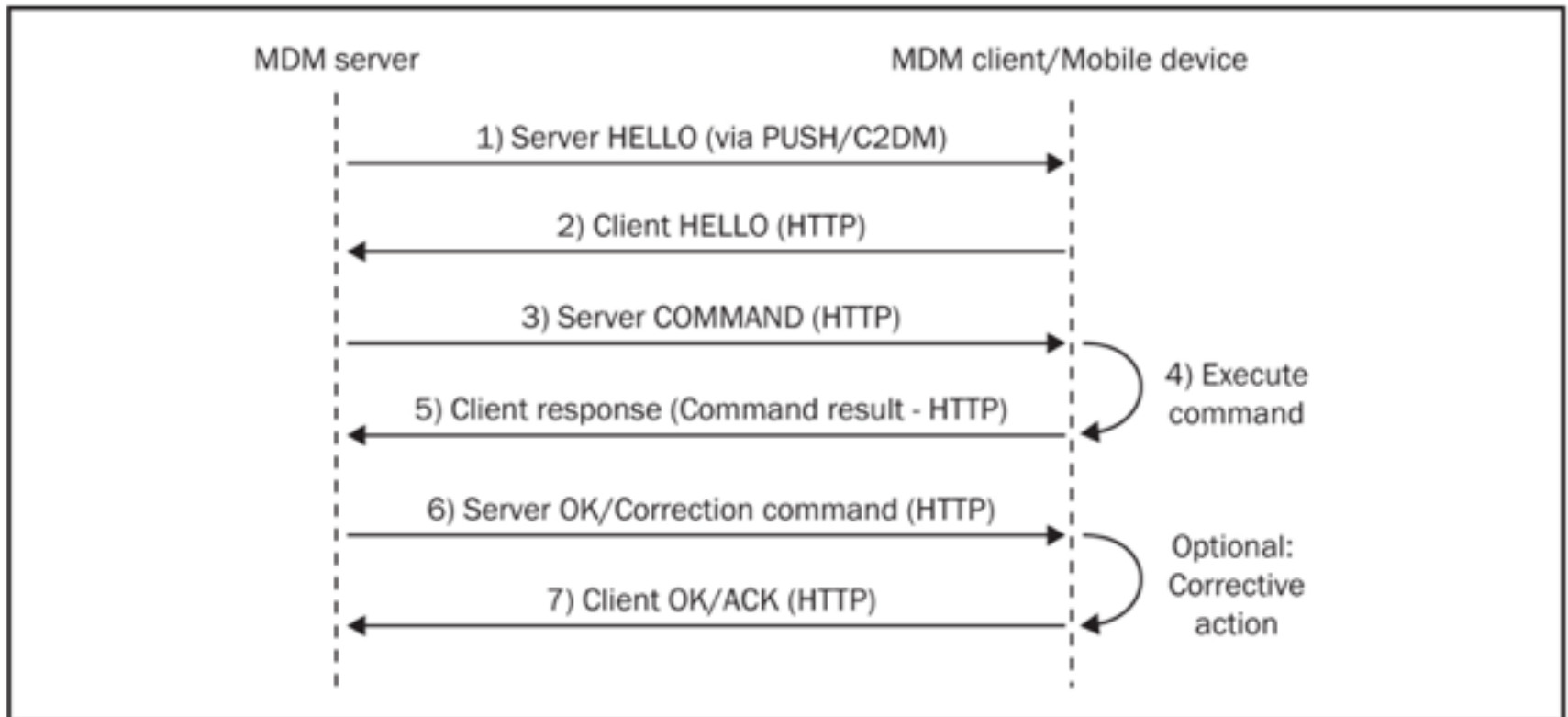
# Detecting MDM Bypass

- End-user can modify the provisioning profiles and system files
- Administrators need to detect those changes
- MDM client apps poll mobile devices to monitor their security posture (Check-In)
- If a device is out of compliance, MDM server can
  - Remote wipe, remote lock, remote locate, etc.

# Weak MDM Bypass Detection

- Some leading MDM solutions only monitored provisioning files
- Changes made to system files were undetected by the back-end servers
  - Even though the devices were sending data indicating noncompliance
- This was patched

# MDM Server-Client Interaction



# App Patching and Modification Attacks

- APK file can be decompiled to Smali code and modified
- Android accepts self-signed certificates
- Attacker can execute the PackageManager command at a Linux shell to install or uninstall packages silently

# Smali Code for Authentication

```
.method public getAuthToken()Ljava/lang/String;
    .locals 1

    .prologue
    .line 80
    iget-object v0, p0, Lcom/google/android/apps/chrome/AccountManagerContainer;->mAuthToken:Ljava/lang/String;

    return-object v0
.end method

.method public getAuthToken(Landroid/accounts/Account;Landroid/accounts/AccountManagerCallback;)V
    .locals 7
    .parameter "account"
    .parameter
    .annotation system Ldalvik/annotation/Signature;
        value = {
            "(",
            "Landroid/accounts/Account;",
            "Landroid/accounts/AccountManagerCallback",
            "<",
            "Landroid/os/Bundle;",
            ">;)V"
        }
    .end annotation
```



# iOS Modification (on Jailbroken Devices)

- Code can be injected into running processes with MobileSubstrate
- Allows runtime patching of system functions
- Captain Hook and Logo are widely used frameworks that use MobileSubstrate

# XCon

- Prevents MDMs from detecting jailbroken devices
- Patched these functions to fool GOOD

```
GmmDefaults: insecureUserDefaults  
GmmDefaults: secureUserDefaults  
GmmDefaults: objectForKey:OptionJailbreakEnhancementServices  
GmmDefaults: objectForKey:OptionJailbreakEnhancementFork  
GmmDefaults: objectForKey:OptionJailbreakEnhancementKernelState  
GmmDefaults: objectForKey:OptionJailbreakEnhancementDevReadPermission  
GmmDefaults: objectForKey:OptionJailbreakEnhancementURL  
GmmDefaults: objectForKey:NocConnectivityPolicyEnable
```

# Decompiling and Debugging Apps

# Android Reverse Engineering

- APK files can be converted to Smali or even Java code with
  - apktool
  - dex2jar
  - JAD

# Android Code Obfuscation

- ProGuard changes some filenames to provide weak protection
- DashO is much more powerful
- All obfuscation is of limited utility
- A determined attacker can still modify code

# iOS Reverse Engineering

- Apps written in Objective-C
- Compiled into low-level machine language
- Reverse-engineering tools:
  - Class-dump
  - Class-dump-x
  - Class-dump-z
- Scan binary iOS apps to extract interface names

# iOS Class Dump from an MDM App

```
@property(retain, nonatomic) AppStorePolicyManager *appStorePolicyManager; // @synthesize appStorePolicyManager;
- (void)onProvisioned;
- (void)processUserStatusChanged;
- (void)completeWipeDevice;
- (void)wipeDeviceFileSystem;
- (BOOL)wipeNextGPAppInList;
- (BOOL)isRemoteLocked;
- (void)setIsRemoteLocked:(BOOL)arg1;
- (BOOL)disabledDueToDeviceWipe;
- (void)otaPolicyUpdateCheck;
- (void)onOtaPolicyUpdateNotif:(id)arg1;
- (void)onTaskDbNotif:(id)arg1;
- (void)onEmailDbNotif:(id)arg1;
- (void)onOptionsDbNotif:(id)arg1;
- (void)startHighSyncRateCheckTimer;
- (void)stopHighSyncRateCheckTimer;
- (void)checkForHighSyncRate;
- (void)invokePerformInitialContactSync:(id)arg1;
- (void)startContactSyncThreadWaitForInited:(BOOL)arg1;
- (void)terminateContactSyncThreadWaitUntilDone:(BOOL)arg1;
- (void)performInitialContactSync;
- (void)syncComplete;
- (void)gracefulShutdown;
- (void)dealloc;
- (void)saveInt:(int)arg1 inPrefsWithKey:(id)arg2;
- (void)mobileConfigSnoozeTimerFired:(id)arg1;
- (void)actionSheet:(id)arg1 clickedButtonAtIndex:(int)arg2;
- (void>alertView:(id)arg1 clickedButtonAtIndex:(int)arg2;
- (void)start2WayContactSync:(BOOL)arg1;
- (void)enableSyncWithAddressBook:(BOOL)arg1;
```

# iOS Anti-Decompilation Tips

- Move critical logic to low-level Simula-style programming languages such as C++ that don't use message passing
- Ensure more generic naming conventions for publicly exposed interfaces and declarations
- Strip symbols when deploying in XCode
- Use dynamic UI components for handling sensitive data and user input, not global variables (to avoid swizzling attacks)



# Swizzling Attacks

- A way to log all uses of a class
- Works on global objects
  - Link Ch 7c

# iOS Anti-Decompilation Tips

- Put all sensitive app logic in private methods, protocols, or anonymous methods
  - To avoid swizzling attacks
- Anti-tamper techniques and solutions that inject guards and protections in the app
  - Such as EnsureIT by Arxan

The ARXAN logo features the word "ARXAN" in a bold, blue, sans-serif font. A stylized blue and orange graphic element, resembling a double-headed arrow or a cross, is positioned behind the letter 'X'.

**ARXAN**

# EnsureIT<sup>®</sup> Family of Products

**Intelligent Software Protection for  
mobile and embedded apps**

## **EnsureIT Secures Mobile & Embedded Apps**

EnsureIT is a proven software protection solution for hardening mobile and embedded code from attacks. It defends app integrity from the tampering and reverse engineering of code to steal intellectual property, violate DRM robustness, and pirate the software.

# Detecting Jailbreaks

# Jailbreak Detection

- iOS 4.2 can detect jailbreaks in the OS itself
- Other mobile platforms don't
- MDM solutions use local apps to detect jailbreaks
  - Methods and effectiveness varies widely
  - May just detect presence of Cydia

# Jailbreak Detection Bypass

- Look for Cydia
  - Try writing files with `fopen()` to look for specific files, and/or look for `su`
- Bypass:
  1. Scan app binaries for interface names such as `isDeviceJailBroken` and `checkDeviceSecurity`
  2. Dynamically patch those methods with `MobileSubstrate`

# Jailbreak Detection Bypass

- Don't install Cydia
- MDMs responded by detecting
  - su, apt-get, or file-write permissions
- Those detections still use method calls and platform features that can be circumvented by skilled attackers

# Jailbreak Detection Bypass Countermeasures

- Perform multiple checks at different locations in the app code
- Move jailbreak detection logic to C++
- Perform compensatory controls and actions on the server rather than on-device logic
- Implement anti-debugging and/or code obfuscation tools/techniques like Arxan



# Remote Wipe and Lock

# Mobile Wipe and Lock

- Two of the most widely used features in MDM solutions
- Response to a security incident
  - Device lost
  - Device breached
  - Device out of compliance with security policy

# Ways to Prevent Remote Wipe and Lock

- Put device in Airplane Mode
- Patch MDM app so it won't perform remote wipe or lock
  - And send false responses back to the server
- Most MDM solutions are susceptible to patching
  - But modern MDMs can use client-side logic to wipe or lock devices without communicating with the server