Google's Android Setup Process Security

Greig Paul, *University of Strathclyde,* James Irvine, *University of Strathclyde*

***Abstract*—Despite considerable research having been carried out into the security of the open-source Android operating system, the vast majority of Android devices run software significantly deviating from the open source core. While many of these changes are introduced by the original equipment manufacturer (OEM), almost every Android device available for sale also features a suite of Google-provided applications and services, which are not part of the Android Open Source Project (AOSP) code. These applications are installed with system-level privileges, and are effectively an extension of the operating system itself. We monitored the process of setting up an Android device, and have identified a number of design weaknesses in the implementation of a number of Google services features which come pre-installed on virtually every Android device on sale today, which could permit skilled and capable attackers to carry out persistent attacks against Android users.**

Keywords*—*Android, security

I. Introduction

GOOGLE reported in its May 2014 I/O event that there were more than one billion active users of Android. Given the near-ubiquity of Android devices, it is worth bearing in mind that, in addition to the open-source Android operating system, Google also has its own proprietary software installed on most Android devices, often referred to as "Google Apps".

Android devices with access to the Google Play Store are shipped with this suite of Google-provided applications, in addition to the core Android operating system. We carried out an investigation into the security of these applications.

II. Network Interception Techniques

Since almost all of the relevant network traffic pertaining to the setup and ongoing configuration of Android devices is encrypted and authenticated using the HTTPS protocol, it was necessary to configure an Android device such that all inbound and outbound network connections could be monitored. To carry out this monitoring, a Wi-Fi network was created, and the Android device's network settings were manually configured to use a Linux computer running the mitmproxy software as the default network gateway. [1] Mitmproxy generates a new certificate authority (CA) file, which was manually added to the system partition of our Android devices, allowing the proxy to generate certificates believed valid by the operating system. This meant that, for the purposes of these tests, our interception CA certificate was considered by the operating system to be no different in any way to the numerous commercial CAs which are included in the operating system by default.

III. Web Communications Security

HTTPS is a secure communications protocol used by web browsers and other software, to attempt to ensure that communications are only established with the legitimate server that the user is attempting to connect to. HTTPS is widely used in online commerce and banking, and is commonly recognised by the 'padlock' icon in web browsers.

To establish an HTTPS connection, the destination server must present a certificate, signed by a Certificate Authority (CA), starting that the server the user is reaching is genuinely operated by the owner of that internet domain name. A large collection of CAs are pre-loaded into mobile devices and web browsers, and are trusted to only issue certificates to the legitimate owners of domain names.

With around 150 CAs enabled by default on modern Android devices, and all CAs able to issue trusted certificates for all domain names, the compromise of any individual CA would permit the generation of rogue certificates for any domain name, including valuable targets such as 'gmail.com'. Such a rogue certificate can then be used to intercept data which a user accesses or provides to a service. [2]

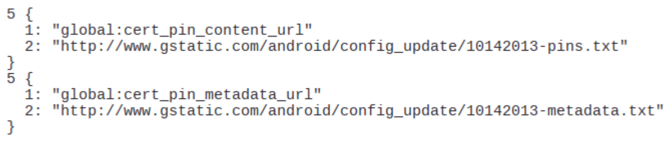
To attempt to prevent such attacks from taking place, a technique known as *certificate pinning* is used, which ensures that only certificates from a particular CA will be trusted for a given server, even if a a valid certificate from another (ordinarily trusted) CA was used. Certificate pinning is implemented in the client software which accesses the HTTPS services, and should contain a record of which domain names will only accept certain certificates.

Support for certificate pinning was introduced in Android version 4.2 [3], although Android does not ship with a list of pre-pinned certificates, even in the most recent Android 4.4 releases, per our findings.

IV. Initial Provisioning

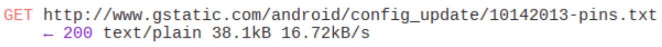
Every newly configured Android device ships with a clean, pre-installed version of the operating system, and undergoes an initial provisioning process when first connected to the internet. This check-in process, as it is commonly referred to, involves the Android device establishing a connection to Google's provisioning servers over HTTP and requesting provisioning data, while at the same time sending device information (including unique device serial numbers such as the device's IMEI number and MAC address). The received provisioning data configures the Android device, setting a number of system-only configuration options within a protected database on the device. [4]

As part of this initial process, a number of URLs are delivered to, and stored on the device, for the purpose of future configuration updates. One of these configuration files pertains to a whitelist of digital certificates for various domain names. The purpose of this list is to increase user security by ensuring that connections to secure websites are not being intercepted due to a man-in-the-middle (MITM) attack, through the compromise of a CA.



```
5 {
  1: "global:cert_pin_content_url"
  2: "http://www.gstatic.com/android/config_update/10142013-pins.txt"
}
5 {
  1: "global:cert_pin_metadata_url"
  2: "http://www.gstatic.com/android/config_update/10142013-metadata.txt"
}
```

Fig. 1. Android check-in data indicating plain HTTP updates



```
GET http://www.gstatic.com/android/config_update/10142013-pins.txt
   ← 200 text/plain 38.1kB 16.72kB/s
```

Fig. 2. GET request over unsecure HTTP for certificate pinning update

The process to update pinned certificates, which is triggered as part of the Google Services Framework, downloads the latest pinning data from Google's servers. While this pinning update data is versioned and signed, to prevent tampering, the actual update process is carried out over a plain, unprotected, HTTP connection, as illustrated in Figure 1.

A. *Certificate Pinning Bypass Attack*

Since a clean install of Android comes with no certificate pins pre-installed, the initial check-in process, carried out over HTTPS, does not benefit from certificate pinning. This means that in the event of a user's connection being compromised, such as in the cases illustrated below, the entire check-in process may be compromised, including the transmission of Google's public key (used for login verification), and the URLs used to carry out future certificate pinning updates.

Even if a user's connection was not compromised at the time of first setup, it is currently possible for a pervasive attacker to block updates to certificate pinning data on an Android device, since the actual transfer is carried out over a plain HTTP connection. Alternatively, such an attacker could replay an existing version of the pinning data (such that the client would not be aware of an updated version being available). This is illustrated in Figure 2. While it is not possible to roll back the pinning data using this update process, the certificate pinning list downloaded at present offers users no protection from rogue certificates - holding a user on the status-quo is beneficial to an attacker.

B. *Present Protection*

The certificate pinning data takes the form of a structured list, detailing the domain name that a pinning record applies to, and the public key (or keys) which are accepted for signing certificates on that domain. There is also a third option, for the enabled state of the pinning entry. Currently, as of August 2014, all of the pinning entries located within the check-in data are disabled, meaning that no certificate pinning is enforced at operating system level. This means that, despite many Google services having certificates listed, none of are actually offering users any protection from man-in-the-middle attacks.

While these pins are disabled (and users are thus not protected from fraudulent servers using different CA certificates), Google are able to receive notifications of any pinning failures through their remote log uploading process referred to internally as the dropbox. It is unclear why these certificate pins are currently disabled. Indeed, we found that there was no pin (even disabled set for 'android.clients.google.com', which is the hostname used for the Google registration and check-in process. As such, under the present configuration, no logs of the setup process being subjected to a man-in-the-middle attack would be uploaded.

This ability for certificate pins to be disabled is documented, and previously discovered [3]. That pinning is not in use for operating system level HTTPS communications does not appear to have previously been identified. This was demonstrated by using certificates signed by our own mock CA without experiencing any errors on the device. As such, we were able to intercept and monitor the entire registration process, using such a compromised connection. For the purposes of this work, our installed CA can be considered to be a regular commercial CA whose certificate was pre-installed on the device.

C. *Going Forward*

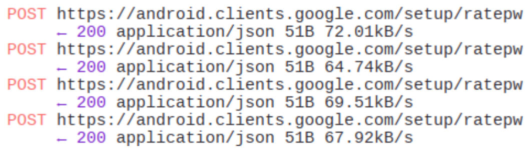
By way of resolution, we propose that Google should ensure that all Android devices ship with a pre-installed list of certificate pins, which would protect the check-in and device setup process. Additionally, the process of updating certificate pinning data should be carried out over an HTTPS connection, to prevent users from having their certificate pinning updates blocked. A pervasive attacker can block a user from receiving any pinning updates through simple filtering of unencrypted packets.

The prospect of the abuse of CA certificates in order to intercept user traffic is not purely theoretical - there have been several documented abuses of CAs in recent years, where users' so-called secure internet traffic was intercepted through the use of compromised CA certificates. [5] [2] The Android operating system remains vulnerable to man-in-the-middle attacks occurring as a result of CA abuse, and users are effectively powerless in preventing this.

Certificate pinning is especially important on mobile platforms, such as Android, particularly within the operating system, where communications with the server are happening 'behind the scenes', such that even a determined user is not able to inspect the details of the connection being used (like they are able to, using a web browser). Such an out-of-sight, out-of-mind approach leaves all users vulnerable to SSL interception attacks, with no practical means for users to detect a different CA being used to sign a certificate. We confirmed, by manually adding a new certificate pin for the check-in server, that certificate pinning prevents the use of an invalid certificate, when enabled.

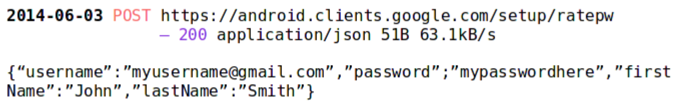
V. Account Registration

During the process of setting up an Android device, the user is invited to either create a new Google account, or log into an existing one. We discovered that during the process of registering a new account, the registration process sends the plaintext contents of the password box, every time its value changes, to Google's servers . While this is sent over HTTPS, the fact we were able to capture this data indicates that an attacker with access to a rogue CA would be able to capture



```
POST https://android.clients.google.com/setup/ratepw
     ← 200 application/json 51B 72.01kB/s
POST https://android.clients.google.com/setup/ratepw
     ← 200 application/json 51B 64.74kB/s
POST https://android.clients.google.com/setup/ratepw
     ← 200 application/json 51B 69.51kB/s
POST https://android.clients.google.com/setup/ratepw
     ← 200 application/json 51B 67.92kB/s
```

Fig. 3. Android setup process sending current password being typed to Google servers



```
2014-06-03 POST https://android.clients.google.com/setup/ratepw
                ← 200 application/json 51B 63.1kB/s

{"username":"myusername@gmail.com","password";"mypasswordhere","first
Name":"John","lastName":"Smith"}
```

Fig. 4. Android setup process sending the plaintext password to the server

this. These recurring requests are shown in Figure 3, which was captured during our research, at the password selection stage of registration.

As is evident from Figure 3, the password data is being sent to Google's servers for real-time password strength rating. Indeed, the server returns a strength rating for the current contents of the password input box, each time it is updated, and this is used to update the user interface.

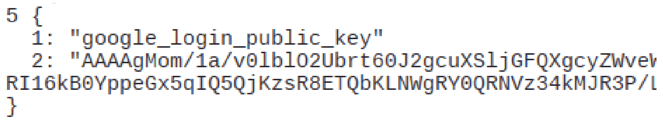
Rather oddly, as shown in Figure 4 the password rating process also sends the proposed email address of the user (which may give side information as to a user's preferred usernames on other sites), as well as their first and last names for the account. We can see no reason whatsoever as to why a simple password security rating process should require such information, nor why the client-side software should offer it to the server. The strength of a password is not in any way related to the identity of the user creating the account, and transmitting supplementary personally identifying information here appears entirely unnecessary.

We also see no reason why the act of measuring password strength should be carried out on a remote server, especially when it involves users transmitting every keystroke they enter in the password field to a remote server in real-time. While client-side validation (where the user's software is trusted to verify input, before it is processed) is obviously not suitable for final validation of submitted values, local password strength estimation could be used in real-time, with server-side validation after a password has been selected.

In light of the goal of a password strength meter being to encourage users to use a stronger password, after they perhaps entered a weaker one, this appears to be a badly designed system, which will result in users inadvertently revealing other, weaker passwords, which they may commonly use. With 55% of adult internet users in the UK admitting that they "use the same password for most, if not all, websites" [6], revealing the plaintext contents of this password entry box, prior to the user selecting a password, poses a significant risk to users, if their connection was being intercepted, as previously discussed.

A. Why Plaintext?

Having demonstrated earlier why simply sending data over HTTPS is not currently sufficient on Android to secure data against a determined and capable attacker, who wishes to launch a man-in-the-middle attack against a user, we were unsure why this password data was firstly being send to a server, and secondly why it was being sent in plaintext. During further research, we uncovered a public key, sent to the device during the check-in process, which appears to be intended for use in encrypting the data used in the login process, as shown in Figure 5.



```
5 {
  1: "google_login_public_key"
  2: "AAAAgMom/1a/v0lblO2Ubrt60J2gcuXSljGFQXgcyZWve
RI16kB0YppeGx5qIQ5QjKzsR8ETQbKLNWgRY0QRNVz34kMJR3P/
}
```

Fig. 5. Google login public key received during device check-in process

Given such a public key is being sent to devices, presumably to allow for the registration process to be carried out with additional asymmetric encryption on top of HTTPS, it is unclear why Google needed to send any such data in plaintext, without prior encryption. After further investigation of the actual login process, we identified at least 2 further occasions where the plaintext password (and other registration data) was submitted to Google's servers, simply over HTTPS, during the account creation process. An encrypted password only appeared to be used after account creation was complete, in the login stage of the process.

In any case, it is clear that Google has access to the actual plaintext password for every user's account. While sending an encrypted password means that Google still would have access to it, this would at least prevent third parties from carrying out attacks on SSL to gain access to the plaintext passwords. Given zero-knowledge login protocols have been publicly disseminated and shared since at least 1989 [7], the transmission of passwords themselves is less than ideal from a security perspective.

B. Going Forward

In order to alleviate this issue, we propose that the password strength meter contained within the registration process should be entirely locally-based, with the code running on the phone itself. This would prevent users from inadvertently divulging passwords they consider using (which may reveal a methodology in their selection of passwords for other services to a determined attacker). With the passwords being sent in plaintext over HTTPS, any man-in-the-middle attacker would be able to view these passwords in transit (like we were, during our experiments).

We also propose that Google should make use of their login public key to encrypt all data transmitted during registration, and that they should extend their extra encryption to cover data other than passwords (such as usernames) during the login process. In the longer term, we propose that one of the many publicly documented zero-knowledge password exchange protocols be used for login and registration, such that Google never places itself in contact with raw user passwords.

VI. Device Security

The Android operating system is built around a security model whereby a device and its data should remain secure, even when in the posession of an attacker (such as someone picking up an unattended phone). For example, the Android Debugging Bridge (ADB) interface requires authentication of the host PC which initiated the request, and such authentication prompts can only be accepted by a device which has been unlocked (using the appropriate PIN or pattern unlock gesture). In order to prevent an attacker from powering off the device and using another (compromised) operating system to access the data, device encryption can be used. This requires the user to enter a PIN or password at each boot, before the operating system loads, in order to decrypt the user data partition on the device.

When device encryption is used on an Android device, the regular screen lock password or PIN is requested at boot. A long-standing request exists on Google's Android Open Source Project (AOSP) issue tracker, where users request that two different and separate passwords be permitted - one for unlocking their screen, and one to decrypt their device. [8] The rationale behind this request is somewhat clear - an average user unlocks their phone 150 times per day, according to research by Kleiner Perkins Caufield Byers. [9] With so many unlock operations per day, the likelihood that someone will be able to shoulder-surf a device unlock code increases significantly.

Convenience is also a significant factor - users will be much more likely to use a short PIN, which is easy for them to enter rapidly, since they need to enter it often. In contrast, any encryption password or key should be longer, such that it contains greater entropy, making brute force attacks against the encryption key impractical. By forcing Android users to use the same PIN or password for unlocking their device, and decrypting its storage, users are therefore more likely to be using weak credentials for their device encryption. As detailed in the enhancement request [8], the underlying technology supports using different passwords for the lockscreen and device encryption, and it would appear such restrictions are not borne out of technical limitation.

A. Device Administrators

Within the Android operating system, device administrator privileges are used to allow third party, non-core software to make use of features which would ordinarily be considered as protected or dangerous. Device administrator privileges can be used to enforce policies on a device, such as requiring a PIN or password be used (or placing requirements on the quality of such passwords), preventing the use of cameras on a device, requiring the use of device encryption, or changing the password currently set on a device.

Ordinarily, device administrator privileges are granted manually by users, who are prompted to allow or deny the request, in a controlled prompt (preventing automated input from interacting with the prompt, or from any kind of on-screen overlay being placed over the prompt to alter its appearance). In the case of the Android Device Manager, however, the device check-in process was able to enable this elevated access, without prompting the user to review and accept the requested device administration privileges.

```
5 {
  1: "mdm.mdm_enabled"
  2: "true"
}
```

Fig. 6. Google check-in process enabling device manager

B. Device Administrators at Check-in

During the check-in process, Android devices obtain configuration information from Google's servers, as discussed previously. One of these configuration parameters pertains to the status of the 'Android Device Manager' feature, which is a part of Google's system application suite, that does not form part of the open source core of Android. This device manager service offers the ability to locate and remotely lock or wipe a lost or stolen Android device, by using the associated Google account.

From our analysis of the data exchanged during the check-in process, we found that the Android Device Manager tool was able to be remotely enabled. On a freshly-reset device, we verified that Android Device Manager was disabled. After connecting the device to the internet, we noted that (without logging into a Google account or similar), the check-in process had enabled device administrator privileges for Android Device Manager. The received request for this is shown in Figure 6.

As it is possible for Google to use the check-in process to remotely enable the device administrator features of Android Device Manager, and this process completes before a user may have even logged into their account, the Android Device Manager feature may pose risks to users during (and after) the setup process. By default, Android Device Manager allows devices to be located and remotely locked or wiped. The process of remote locking, however, is of particular interest, as it exposes a major limitation of the platform. If a user loses their device, they can log into their Google account online and set a lockscreen password, to protect their device in case it is found. In the event that the device already has such a password, it is overridden and replaced with the new one. If the device uses encryption, the data encryption password is also updated at this time.

As a result, it is possible for the encryption password of an internet-connected Android device to be remotely changed, without physical access to the device, with either the cooperation of Google (to send such a request from their servers to the device), or access to the Google account password (to use the Android Device Manager web interface to issue the change password request). As the account registration process may be monitored (and thus compromised) by a pervasive and determined adversary, as discussed previously, users whose connections may be monitored should be aware of this risk.

In our research, we configured a freshly-reset device with a new Google account, and proceeded with the default settings (i.e. without altering any configuration or Android Device Manager options), to simulate the actions of a regular user who is unaware of the existence of such features. After enabling device encryption (to simulate the process of a device being used in an enterprise environment), we were able to repeatedly change the device encryption password, and lockscreen password, over the internet. Encrypted devices containing enterprise data are therefore only as secure as Google and its remote device password reset process.

In the process of our research, we have not identified any documentation which states that the device encryption key may be remotely altered, over an authenticated network connection. The Android source code API documentation for the resetPassword function does not make reference to the fact that the encryption password will be altered when using it, but we have verified that any change (manual or programmatic) to the device PIN or password will be propogated to the encryption password.

In light of our previous discoveries, however, we believe that this poses a significant risk to users - with the disclosure of plaintext passwords during registration, and the very limited protection against man-in-the-middle attacks against SSL, we believe it practical for a determined attacker to subvert the encryption present on an Android device, either through the cooperation or coercion of Google, or by obtaining the user's own account credentials, and carrying out the remote operation to change the encryption key. In any case, even without such information available, due to the significant trade-off between usability and security, it is likely that the majority of users are making use of a fairly weak device encryption password, in order to aid their entering of that password every time they unlock the device.

VII. Security in Core Android?

The above weaknesses in Android have been considered from the perspective of a user of an Android device which ships with additional closed-source Google services pre-installed. A natural reaction may be for concerned users to consider using Android without such services being installed on their device. From our research, we have found that the Android check-in process is not included on devices which solely run the open source components of Android (distributed as the Android Open Source Project (AOSP) for developers to compile themselves) [10]. There is also no Google login or registration service present, and there is no Android Device Manager present on such devices.

As such, a device running AOSP, without any Google services, should not be vulnerable to such attacks. Nonetheless, such devices are themselves not offered protection by way of certificate pinning, since Android ships with an empty pinning list, and requires Google services to update pinning data. As such, while the core open-source components of the Android operating system do themselves support the use of certificate pinning, a user would receive no protection, as they would not receive any certificate pins over the internet, to be used.

Google's proprietary Chrome browser features its own certificate pinning mechanisms, which operate independently of the system-based pinning, and as such offer users the same certificate pinning features as users of its desktop browser. During our research, we found that while the operating system was accepting of our own certificates (which were injected onto the system partition, such that they were treated as any other legitimate CA certificate), the Chrome browser would correctly detect and prevent use of any HTTPS services which used certificate pinning. Users of the open source browser present in AOSP are thus not offered the same level of protection as those using the closed-source Chrome browser.

Since the operating system certificate handling is used by the developer-ready webview control, third party applications which implement their own web browsers or web views are equally unprotected at present.

VIII. Conclusion

We have identified three main weaknesses within near-universally distributed components of the Android platform, which do not form part of the core open-source operating system. The first of these weaknesses pertains to the currently-inoperative certificate pinning implementation, which uses Google's proprietary services framework to download certificate pin lists over a plain HTTP connection, which is vulnerable to replay or blocking attacks, to prevent users from receiving updated certificate pinning data. The current certificate pinning data does not offer any protection to users, as enforcement of all certificate pins is currently disabled, despite the Android 4.4 release notes stating users are protected from connecting to forged Google servers via pinning.

The second weakness relates to the Google account registration process, presented to users upon running a new phone. When registering a new account, the contents of the password box (as well as their name and username) is sent to Google over an HTTPS connection, in plaintext, every time a user alters its contents, apparently for the purpose of verifying password strength. We remain unsure as to why this is done server-side, and highlight the risks that this poses for users who re-use passwords, who may find they try several passwords before selecting one that is suitably strong, as they would have disclosed several of their passwords used elsewhere in the process.

The final weakness pertains to the implementation of Android's device encryption, which makes it possible for someone with knowledge of a Google account password (or an attacker able to socially engineer or compel Google) to reset the device encryption password, as well as the lockscreen password of an Android device which runs the Android Device Manager service (which is able to be remotely enabled using the Google check-in process). This poses a risk to enterprise users of devices, who may not be aware of such abilities being enabled on devices by default by Google.

Acknowledgment

This work was funded by EPSRC Doctoral Training Grant EP/K503174/1.

References

[1] A. Cortesi. mitmproxy: a man-in-the-middle proxy. [Online]. Available: http://mitmproxy.org

[2] (2011, September) Iranians hit in email hack attack. BBC News. [Online]. Available: http://www.bbc.co.uk/news/technology-14802673

[3] N. Elenkov. (2012, December) Certificate pinning in android 4.2. [Online]. Available: http://nelenkov.blogspot.co.uk/2012/12/certificate-pinning-in-android-42.html

[4] L. Øverlier, “Data leakage from android smartphones,” Ph.D. dissertation, Masters thesis, Norwegian Defence Research Establishment (FFI), 2012.

[5] M. Lee. (2013, December) Google catches french govt spoofing its domain certificates. Google Inc. [Online]. Available: http://www.zdnet.com/google-catches-french-govt-spoofing-its-domain-certificates-7000024062/

[6] (2013, April) UK adults taking online password security risks. Ofcom. [Online]. Available: http://media.ofcom.org.uk/news/2013/uk-adults-taking-online-password-security-risks/

[7] C. I. Jaramillo, G. Richard, S. Sperry, and W. Patterson, “An implementation of a zero-knowledge protocol for a secure network login procedure,” in *Southeastcon’89. Proceedings. Energy and Information Technologies in the Southeast., IEEE.* IEEE, 1989, pp. 197–201.

[8] (2012, April) Android issue tracker - different passwords for encryption and screen lock. Google Inc. [Online]. Available: https://code.google.com/p/android/issues/detail?id=29468

[9] M. Meeker and L. Wu, “Internet trends D11 conference,” 2013.

[10] (2013, September) Welcome to the Android open source project. Google Inc. [Online]. Available: http://source.android.com/