Why Eve and Mallory Love Android: An Analysis of Android SSL (In)Security

Sascha Fahl, Marian Harbach, Thomas Muders, Matthew Smith
Distributed Computing & Security Group
Leibniz University of Hannover
Hannover, Germany
{fahl,harbach,muders,smith}@dcsec.uni-hannover.de

Lars Baumgärtner, Bernd Freisleben
Department of Math. & Computer Science
Philipps University of Marburg
Marburg, Germany
{lbaumgaertner, freisleb}@informatik.uni-marburg.de

ABSTRACT

Many Android apps have a legitimate need to communicate over the Internet and are then responsible for protecting potentially sensitive data during transit. This paper seeks to better understand the potential security threats posed by benign Android apps that use the SSL/TLS protocols to protect data they transmit. Since the lack of visual security indicators for SSL/TLS usage and the inadequate use of SSL/TLS can be exploited to launch Man-in-the-Middle (MITM) attacks, an analysis of 13,500 popular free apps downloaded from Google's Play Market is presented.

We introduce MalloDroid, a tool to detect potential vulnerability against MITM attacks. Our analysis revealed that 1,074 (8.0%) of the apps examined contain SSL/TLS code that is potentially vulnerable to MITM attacks. Various forms of SSL/TLS misuse were discovered during a further manual audit of 100 selected apps that allowed us to successfully launch MITM attacks against 41 apps and gather a large variety of sensitive data. Furthermore, an online survey was conducted to evaluate users' perceptions of certificate warnings and HTTPS visual security indicators in Android's browser, showing that half of the 754 participating users were not able to correctly judge whether their browser session was protected by SSL/TLS or not. We conclude by considering the implications of these findings and discuss several countermeasures with which these problems could be alleviated.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Network**]: Network Operations—*Public Networks*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Data Sharing*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.
Copyright 2012 ACM 978-1-4503-1651-4/12/10 ...$10.00.

General Terms

Security, Human Factors

Keywords

Android, Security, Apps, MITMA, SSL

1. INTRODUCTION

Currently, Android is the most used smartphone operating system in the world, with a market share of 48%[1] and over 400,000 applications (apps) available in the Google Play Market[2], almost doubling the number of apps in only six months.[3] Android apps have been installed over 10 billion times [4] and cover a vast range of categories from games and entertainment to financial and business services. Unlike the "walled garden" approach of Apple's App Store, Android software development and the Google Play Market are relatively open and unrestricted. This offers both developers and users more flexibility and freedom, but also creates significant security challenges.

The coarse permission system [9] and over-privileging of applications [16] can lead to exploitable applications. Consequently, several efforts have been made to investigate privilege problems in Android apps [17, 9, 4, 3, 18]. Enck et al. introduced TaintDroid [7] to track privacy-related information flows to discover such (semi-)malicious apps. Bugiel et al. [3] showed that colluding malicious apps can facilitate information leakage. Furthermore, Enck et al. analyzed 1,100 Android apps for malicious activity and detected widespread use of privacy-related information such as IMEI, IMSI, and ICC-ID for "cookie-esque" tracking. However, no other malicious activities were found, in particular no exploitable vulnerabilities that could have lead to malicious control of a smartphone were observed [8].

In this paper, instead of focusing on malicious apps, we investigate potential security threats posed by benign Android apps that legitimately process privacy-related user data, such as log-in credentials, personal documents, contacts, financial data, messages, pictures or videos. Many of these apps communicate over the Internet for legitimate reasons and thus request and require the INTERNET permission. It is then

[1] `http://android-ssl.org/s/1`
[2] `http://android-ssl.org/s/2`
[3] `http://android-ssl.org/s/3`
[4] `http://android-ssl.org/s/4`

necessary to trust that the app adequately protects sensitive data when transmitting via the Internet.

The most common approach to protect data during communication on the Android platform is to use the Secure Sockets Layer (SSL) or Transport Layer Security (TLS) protocols.[5] To evaluate the state of SSL use in Android apps, we downloaded 13,500 popular free apps from Google's Play Market and studied their properties with respect to the usage of SSL. In particular, we analyzed the apps' vulnerabilities against Man-in-the-Middle (MITM) attacks due to the inadequate or incorrect use of SSL.

For this purpose, we created MalloDroid, an Androguard[6] extension that performs static code analysis to a) analyze the networking API calls and extract valid HTTP(S) URLs from the decompiled apps; b) check the validity of the SSL certificates of all extracted HTTPS hosts; and c) identify apps that contain API calls that differ from Android's default SSL usage, e. g., contain non-default trust managers, SSL socket factories or hostname verifiers with permissive verification strategies. Based on the results of the static code analysis, we selected 100 apps for manual audit to investigate various forms of SSL use and misuse: accepting all SSL certificates, allowing all hostnames regardless of the certificate's Common Name (CN), neglecting precautions against SSL stripping, trusting all available Certificate Authorities (CAs), not using SSL pinning, and misinforming users about SSL usage.

Furthermore, we studied the visibility and awareness of SSL security in the context of Android apps. In Android, the user of an app has no guarantee that an app uses SSL and also gets no feedback from the Android operating system whether SSL is used during communication or not. It is entirely up to the app to use SSL and to (mis)inform the user about the security of the connection. However, even when apps present warnings and security indicators, users need to see and interpret them correctly. The users' perceptions concerning these warnings and indicators were investigated in an online survey. Finally, several countermeasures that could help to alleviate the problems discovered in the course of our work are discussed.

The results of our investigations can be summarized as follows:

- 1,074 apps contain SSL specific code that either accepts all certificates or all hostnames for a certificate and thus are potentially vulnerable to MITM attacks.
- 41 of the 100 apps selected for manual audit were vulnerable to MITM attacks due to various forms of SSL misuse.
- The cumulative install base of the apps with confirmed vulnerabilities against MITM attacks lies between 39.5 and 185 million users, according to Google's Play Market.[7] This number includes 3 apps with install bases between 10 and 50 million users each.
- From these 41 apps, we were able to capture credentials for American Express, Diners Club, Paypal, bank accounts, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, remote control servers, arbitrary email accounts, and IBM Sametime, among others.
- We were able to inject virus signatures into an antivirus app to detect arbitrary apps as a virus or disable virus detection completely.
- It was possible to remotely inject and execute code in an app created by a vulnerable app-building framework.
- 378 (50.1%) of the 754 Android users participating in the online survey did not judge the security state of a browser session correctly.
- 419 (55.6%) of the 754 participants had not seen a certificate warning before and typically rated the risk they were warned against as medium to low.

The paper is organized as follows. Section 2 gives background information on how SSL is used in Android and how MITM attacks can be launched. Section 3 discusses related work. In Section 4, the usage of SSL in 13,500 Android apps is investigated using MalloDroid. Section 5 presents the results of manual audits of 100 selected apps to determine what kind of data is actually sent via the possibly broken SSL communication channels. The limitations of our app analysis are discussed in Section 6. Section 7 describes the results of our online survey to investigate the users' perceptions concerning certificate warnings and secure connections in their Android browsers. In Section 8, possible countermeasures against unencrypted traffic and SSL misuse are discussed. Section 9 concludes the paper and outlines directions for future work.

2. BACKGROUND

The focus of our investigation is the inadequate use of SSL in Android apps. In this section, we give a brief overview of how SSL is used in Android and how MITM attacks can be launched against broken SSL connections in the context of this paper.

2.1 SSL

The Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), are cryptographic protocols that were introduced to protect network communication from eavesdropping and tampering. To establish a secure connection, a client must securely gain access to the public key of the server. In most client/server setups, the server obtains an X.509 certificate that contains the server's public key and is signed by a Certificate Authority (CA). When the client connects to the server, the certificate is transferred to the client. The client must then validate the certificate [2]. However, validation checks are not a central part of the SSL and X.509 standards. Recommendations are given, but the actual implementation is left to the application developer.

The basic validation checks include: a) does the subject (CN) of the certificate match the destination selected by the client?; b) is the signing CA a trusted CA?; c) is the signature correct?; and d) is the certificate valid in terms of

[5]Android supports both SSL and TLS; for brevity, we will refer to both protocols as SSL. The issues described in this paper affect both SSL and TLS in the same way.

[6]`http://code.google.com/p/androguard/`

[7]Google's Play Market only gives a range within which the number of installed apps lies based on the installs from the Play Market. The actual number is likely to be larger, since alternative app markets for Android also contribute to the install base.

its time of expiry? Additionally, revocation of a certificate and its corresponding certificate chain should be checked, but downloading Certificate Revocation Lists (CRLs) or using the Online Certificate Status Protocol (OCSP, [1]) is often omitted. The open nature of the standard specification has several pitfalls, both on a technical and a human level. Therefore, our evaluations in the remainder of this paper are based on examining the four validation checks listed above.

2.2 Android & SSL

The Android 4.0 SDK offers several convenient ways to access the network. The `java.net`, `javax.net`, `android.net` and `org.apache.http` packages can be used to create (server) sockets or HTTP(S) connections. The `org.webkit` package provides access to web browser functionality. In general, Android allows apps to customize SSL usage – i. e., developers must ensure that they use SSL correctly for the intended usage and threat environment. Hence, the following (mis-)use cases can arise and can cause an app to transmit sensitive information over a potentially broken SSL channel:

Trusting all Certificates. The TrustManager interface can be implemented to trust all certificates, irrespective of who signed them or even for what subject they were issued.

Allowing all Hostnames. It is possible to forgo checks of whether the certificate was issued for this address or not, i. e., when accessing the server `example.com`, a certificate issued for `some-other-domain.com` is accepted.

Trusting many CAs. This is not necessarily a flaw, but Android 4.0 trusts 134 CA root certificates per default. Due to the attacks on several CAs in 2011, the problem of the large number of trusted CAs is actively debated.[8]

Mixed-Mode/No SSL. App developers are free to mix secure and insecure connections in the same app or not use SSL at all. This is not directly an SSL issue, but it is relevant to mention that there are no outward signs and no possibility for a common app user to check whether a secure connection is being used. This opens the door for attacks such as SSL stripping [12, 13] or tools like Firesheep.[9]

On the other hand, Android's flexibility in terms of SSL handling allows advanced features to be implemented. One important example is SSL Pinning[10], in which either a (smaller) custom list of trusted CAs or even a custom list of specific certificates is used. Android does not offer SSL pinning capabilities out of the box. However, it is possible to create a custom trust manager to implement SSL pinning.[11]

The use of an SSL channel, even under the conditions described above, is still more secure than using only plain HTTP against a passive attacker. An active MITM attack is required for an attacker to subvert an SSL channel and is described below.

[8] `http://android-ssl.org/s/5`
[9] `https://codebutler.com/firesheep`
[10] `http://android-ssl.org/s/6`
[11] `http://android-ssl.org/s/7`

2.3 MITM Attack

In a MITM attack (MITMA), the attacker is in a position to intercept messages sent between communication partners. In a passive MITMA, the attacker can only eavesdrop on the communication (attacker label: Eve), and in an active MITMA, the attacker can also tamper with the communication (attacker label: Mallory). MITMAs against mobile devices are somewhat easier to execute than against traditional desktop computers, since the use of mobile devices frequently occurs in changing and untrusted environments. Specifically, the use of open access points [11] and the evil twin attack [21] make MITMAs against mobile devices a serious threat.

SSL is fundamentally capable of preventing both Eve and Mallory from executing their attacks. However, the cases described above open up attack vectors for both Eve and Mallory. Trivially, the mixed-mode/no SSL case allows Eve to eavesdrop on non-protected communication.

SSL stripping is another method by which a MITMA can be launched against an SSL connection, exploiting apps that use a mix of HTTP and HTTPS. SSL stripping relies on the fact that many SSL connections are established by clicking on a link in or being redirected from a non-SSL-protected site. During SSL stripping, Mallory replaces *https://* links in the non-protected sites with insecure *http://* links. Thus, unless the user notices that the links have been tampered with, Mallory can circumvent SSL protection altogether. This attack is mainly relevant to browser apps or apps using Android's WebView.

3. RELATED WORK

To the best of our knowledge, there is no in-depth study of SSL usage and security on Android phones to date. Thus, the discussion of related work is divided into two parts: related work concerning Android security and a selection of SSL security work relevant for this paper.

3.1 Android Security

There have been several efforts to investigate Android permissions and unwanted or malicious information flows, such as the work presented by Enck et al. [9, 7], Porter Felt et al. [17, 16], Davi et al. [4], Bugiel et al. [3], Nauman et al. [15] and Egners et al. [6]. While these papers show how permissions can be abused and how this abuse can be prevented, their scope does not include the study of SSL issues, and the proposed countermeasures do not mitigate the threats presented in this paper. We present vulnerabilities based on weaknesses in the design and use of SSL and HTTPS in Android apps. Since the permissions used by the apps during SSL connection establishment are legitimate and necessary, the current permissions-based countermeasures would not help.

There are several good overviews of the Android security model and threat landscape, such as Vidas et al. [25], Shabatai [19] et al. and Enck et al. [10, 8]. These papers do not discuss the vulnerability of SSL or HTTPS on Android. Enck et al. [8] does mention that some apps use sockets directly, bearing the potential for vulnerabilities, but no malicious use was found (cf. [8], Finding 13). Our investigation shows that there are several SSL-related vulnerabilities in Android apps, endangering millions of users.

McDaniel et al. [14] and Zhou et al. [26] also mainly focus on malicious apps in their work on the security issues

associated with the app market model of software deployment. The heuristics of DroidRanger [26] could be extended to detect the vulnerabilities uncovered in our work.

3.2 SSL Security

A good overview of current SSL problems can be found in Marlinspike's Black Hat talks [12, 13]. The talks cover issues of security indicators, Common Name (CN) mismatches and the large number of trusted CAs and intermediate CAs. Marlinspike also introduces the SSL stripping attack. The fact that many HTTPS connections are initiated by clicking a link or via redirects is particularly relevant for mobile devices, since the MITMA needed for SSL stripping is easier to execute [21, 11] and the visual indicators are hard to see on mobile devices.

Shin et al. [20] study the problem of SSL stripping for desktop browsers and present a visual-security-cue-based approach to hinder SSL stripping in this environment. They also highlight the particular problem of this type of attack in the mobile environment and suggest that it should be studied in more detail.

Egelman et al. [5] and Sunshine et al. [24] both study the effectiveness of browser warnings, showing that their effectiveness is limited and that there are significant usability issues. Although both of these studies were conducted in a desktop environment, the same caveats need to be considered for mobile devices. In this paper, we conduct a first online survey to gauge the awareness and effectiveness of browser certificate warnings and HTTPS visual security indicators on Android.

4. EVALUATING ANDROID SSL USAGE

Our study of Android SSL security encompasses popular free apps from Google's Play Market. Overall, we investigated 13,500 applications. We built MalloDroid, an extension of the Androguard reverse engineering framework, to automatically perform the following steps of static code analysis:

Permissions. MalloDroid checks which apps *request the INTERNET permission*, which apps *actually contain INTERNET permission-related API calls* and which apps *additionally request and use privacy-related permissions* (cf. [16]).

Networking API Calls. MalloDroid analyzes the use of *HTTP transport* and *Non-HTTP transport* (e. g., direct socket connections).

HTTP vs HTTPS. MalloDroid checks the validity of URLs found in apps and groups the apps into *HTTP only*, *mixed-mode (HTTP and HTTPS)* and *HTTPS only*.

HTTPS Available. MalloDroid tries to establish a secure connection to HTTP URLs found in apps.

Deployed Certificates. MalloDroid downloads and evaluates SSL certificates of hosts referenced in apps.

SSL Validation. MalloDroid examines apps with respect to inadequate SSL validation (e. g., apps containing code that allows all hostnames or accepts all certificates).

12,534 (92.84%) of the apps in our test set request the network permission `android.permission.INTERNET`. 11,938 (88.42%) apps actually perform networking related API calls. 6,907 (51.16%) of the apps in our sample use the INTERNET permission in addition to permissions to access privacy related information such as the users' calendars, contacts, browser histories, profile information, social streams, short messages or exact geographic locations. This subset of apps has the potential to transfer privacy-related information via the Internet. This subset does not include apps such as banking, business, email, social networking or instant messaging apps that intrinsically contain privacy-relevant information without requiring additional permissions.

We found that 91.7% of all networking API calls are related to HTTP(S). Therefore, we decided to focus our further analysis on the usage of HTTP(S). To find out whether an app communicates via HTTP, HTTPS, or both, MalloDroid analyzes HTTP(S) specific API calls and extracts URLs from the decompiled apps.

4.1 HTTP vs. HTTPS

MalloDroid extracted 254,022 URLs. It can be configured to remove certain types of URLs for specific analysis. For this study, we removed 58,617 URLs pointing to namespace descriptors and images, since these typically are not used to transmit sensitive user information. The remaining 195,405 URLs pointed to 25,975 unique hosts. 29,685 of the URLs (15.2%) pointing to 1,725 unique hosts (6.6%) are HTTPS URLs. We further analyzed how many of the hosts referenced in HTTP URLs could also have been accessed using HTTPS.

76,435 URLs (39.1%) pointing to 4,526 hosts (17.4%) allowed a valid HTTPS connection to be established, using Android's default trust roots and validation behavior of current browsers. This means that 9,934 (73.6%) of all 13,500 tested apps could have used HTTPS instead of HTTP with minimal effort by adding a single character to the target URLs. We found that 6,214 (46.0%) of the apps contain HTTPS and HTTP URLs simultaneously and 5,810 (43.0%) do not contain HTTPS URLs at all. Only 111 apps (0.8%) exclusively contained HTTPS URLs.

For a more detailed investigation, we looked at the top 50 hosts, ranked by the number of occurrences. This group mainly consists of advertising companies and social networking sites. These two categories account for 37.9% of the total URLs found, and the hosts are contained in 9,815 (78.3%) of the apps that request the INTERNET permission.

Table 1 presents an overview of the top 10 hosts. The URLs pointing to the these hosts suggest they are often used for Web Service API calls, authentication and fetching/sending user or app information. Especially in the case of ad networks that collect phone identifiers and geolocations [7] and social networks that transport user-generated content, the contained information is potentially sensitive.

34 of the top 50 hosts offer all their API calls via HTTPS, but none is accessed exclusively via HTTPS. Of all the URLs pointing to the top 50 hosts, 22.1% used HTTPS, 61.0% could have used HTTPS by substituting *http://* with *https://*, and 16.9% had to use HTTP because HTTPS was not available. The hosts `facebook.com` and `tapjoyads.com` are positive examples, since the majority of the URLs we found for these two hosts already use HTTPS.

Table 1: The top 10 hosts used in all extracted URLs and their SSL availability, total number of URLs and number of HTTPS URLs pointing to that host.

Host	has SSL	# URLs	# HTTPS
market.android.com	✓	6,254	3,217
api.airpush.com	✓	5,551	0
a.admob.com	✓	4,299	0
ws.tapjoyads.com	✓	3,410	3,399
api.twitter.com	✓	3,220	768
data.flurry.com	✓	3,156	1,578
data.mobclix.com	✓	2,975	0
ad.flurry.com	✓	2,550	0
twitter.com	✓	2,410	129
graph.facebook.com	✓	2,141	1,941

4.2 Deployed SSL Certificates

To analyze the validity of the certificates used by HTTPS hosts, we downloaded the SSL certificates for all HTTPS hosts extracted from our app test set, yielding 1,887 unique SSL certificates. Of these certificates, 162 (8.59%) failed the verification of Android's default SSL certificate verification strategies, i. e., 668 apps contain HTTPS URLs pointing to hosts with certificates that could not be validated with the default strategies. 42 (2.22%) of these certificates failed SSL verification because they were self-signed, i. e., HTTPS links to self-signed certificates are included in 271 apps. 21 (1.11%) of these certificates were already expired, i. e., 43 apps contain HTTPS links to hosts with expired SSL certificates.

For hostname verification, we applied two different strategies that are also available in Android: the *BrowserCompatHostnameVerifier*[12] and the *StrictHostnameVerifier*[13] strategy. We found 112 (5.94%) certificates that did not pass strict hostname verification, of which 100 certificates also did not pass the browser compatible hostname verification. Mapping these certificates to apps revealed that 332 apps contained HTTPS URLs with hostnames failing the *BrowserCompatHostnameVerifier* strategy.

Overall, 142 authorities signed 1,887 certificates. For 45 (2.38%) certificates, no valid certification paths could be found, i. e., these certificates were signed by authorities not reachable via the default trust anchors. These certificates are used by 46 apps. All in all, 394 apps include HTTPS URLs for hosts that have certificates that are either expired, self-signed, have mismatching CNs or are signed by non-default-trusted CAs.

4.3 Custom SSL Validation

Using MalloDroid, we found 1,074 apps (17.28% of all apps that contain HTTPS URLs) that include code that either bypasses effective SSL verification completely by accepting all certificates (790 apps) or that contain code that accepts all hostnames for a certificate as long as a trusted CA signed the certificate (284 apps).

While an app developer wishing to accept all SSL certificates must implement the TrustManager interface and/or extend the SSLSocketFactory class, allowing all hostnames only requires the use of the `org.apache.http.conn.ssl.AllowAllHostnameVerifier` that is included in 453 apps. Additionally, MalloDroid found a `FakeHostnameVerifier`, `NaiveHostnameVerifier` and `AcceptAllHostnameVerifier` class that can be used in the same way.

To understand how apps use "customized" SSL implementations, we searched for apps that contain non-default trust managers, SSL socket factories and hostname verifiers differing from the `BrowserCompatHostnameVerifier` strategy. We found 86 custom trust managers and SSL socket factories in 878 apps. More critically, our analysis also discovered 22 classes implementing the `TrustManager` interface and 16 classes extending the `SSLSocketFactory` that accept all SSL certificates. Table 2 shows which broken trust managers and SSL socket factories were found.

Table 2: Trust Managers & Socket Factories that trust all certificates (suffixes omitted to fit the page)

Trust Managers	SSL Socket Factories
`AcceptAllTrustM`	`AcceptAllSSLSocketF`
`AllTrustM`	`AllTrustingSSLSocketF`
`DummyTrustM`	`AllTrustSSLSocketF`
`EasyX509TrustM`	`AllSSLSocketF`
`FakeTrustM`	`DummySSLSocketF`
`FakeX509TrustM`	`EasySSLSocketF`
`FullX509TrustM`	`FakeSSLSocketF`
`NaiveTrustM`	`InsecureSSLSocketF`
`NonValidatingTrustM`	`NonValidatingSSLSocketF`
`NullTrustM`	`NaiveSslSocketF`
`OpenTrustM`	`SimpleSSLSocketF`
`PermissiveX509TrustM`	`SSLSocketFUntrustedCert`
`SimpleTrustM`	`SSLUntrustedSocketF`
`SimpleX509TrustM`	`TrustAllSSLSocketF`
`TrivialTrustM`	`TrustEveryoneSocketF`
`TrustAllManager`	`NaiveTrustManagerF`
`TrustAllTrustM`	`LazySSLSocketF`
`TrustAnyCertTrustM`	`UnsecureTrustManagerF`
`UnsafeX509TrustM`	
`VoidTrustM`	

This small number of critical classes affects a large number of apps. Many of the above classes belong to libraries and frameworks that are used by many apps. 313 apps contained calls to the `NaiveTrustManager` class that is provided by a crash report library. In 90 apps, MalloDroid found the `NonValidatingTrustManager` class provided by an SDK for developing mobile apps for different platforms with just a single codebase. The `PermissiveX509TrustManager`, found in a library for sending different kinds of push notifications to Android devices, is included in 76 apps. Finally, in 78 apps, MalloDroid found a `SSLSocketFactory` provided by a developer library that accepts all certificates. The library is intended to support developers to write well designed software and promotes itself as a library for super-easy and robust networking. Using any of the above Trust Managers or Socket Factories results in the app trusting all certificates.

5. MITMA STUDY

The static code analysis presented above only shows the potential for security problems. The fact that code for insecure SSL is present in an app does not necessarily mean that it is used or that sensitive information is passed along it. Even more detailed automated code analysis, such as control flow analysis, data flow analysis, structural analysis and semantic analysis cannot guarantee that all uses are correctly

[12] `http://android-ssl.org/s/8`
[13] `http://android-ssl.org/s/9`

identified [8]. Thus, we decided to conduct a more detailed manual study to find out what sort of information is actually sent via these potentially broken SSL communication channels, by installing apps on a real phone and executing an active MITMA against the apps. For this part of the study, we narrowed our search down to apps from the Finance, Business, Communication, Social and Tools categories, where we suspected a higher amount of privacy relevant information and a higher motivation to protect the information. In this test set, there are 266 apps containing broken SSL or hostname verifiers (Finance: 45, Social: 94, Communication: 49, Business: 60, Tools: 18). We ranked these apps based on their number of downloads and selected the top 100 apps for manual auditing. Additionally, we cherry-picked 10 high profile apps (large install base, popular services) that contained no SSL-related API calls but contained potentially sensitive information, to see whether this information was actually sent in the clear or whether some protection mechanism other than SSL was involved.

5.1 Test Environment

For the manual app auditing, we used a Samsung Galaxy Nexus smartphone with Android 4.0 Ice Cream Sandwich. We installed the potentially vulnerable apps on the phone and set up a WiFi access point with a MITM SSL proxy. Depending on the vulnerability to be examined, we equipped the SSL proxy either with a self-signed certificate or with one that was signed by a trusted CA, but for an unrelated hostname.

Of the 100 apps selected for manual audit, 41 apps proved to have exploitable vulnerabilities. We could gather bank account information, payment credentials for PayPal, American Express and others. Furthermore, Facebook, email and cloud storage credentials and messages were leaked, access to IP cameras was gained and control channels for apps and remote servers could be subverted. According to Google's Play Market, the combined install base of the vulnerable apps in our test set of 100 apps was between 39.5 and 185 million users at the time of writing. In the following, we briefly discuss our findings to illustrate the scope of the problem.

5.2 Trusting All Certificates

21 apps among the 100 selected apps were vulnerable to this attack. We gave our MITMA proxy a self-signed certificate for the attack. The apps leaked information such as login credentials, webcam access or banking data. One noteworthy contender was a generic online banking app. The app uses separate classes for each bank containing different trust manager implementations. 24 of the 43 banks supported were not protected from our MITMA. The app also leaks login credentials for American Express, Diners Club and Paypal. The Google Play Market reports an install base between 100,000 and half a million users. A further app vulnerable to this attack offers instant messaging for the Windows Live Messenger service. The app has an install base of 10 to 50 million users and is listed in the top 20 apps for the communication category in the Google Play Market (as of April 30th, 2012). Username and password are both sent via a broken SSL channel and were sniffed during our attack. This effectively gives an attacker full access to a Windows Live account that can be used for email, messaging or Microsoft's SkyDrive cloud storage. We also found a browser with an install base between 500,000 and one million users that trusts all certificates. The browser does not correctly handle SSL at all, i. e., it accepts an arbitrary certificate for every website the user visits and hence leaks whatever data the user enters. All three apps do not provide any SSL control or configuration options for the user. None of the other apps vulnerable to this attack showed warning messages to the user while the MITMA was being executed.

5.3 Allowing All Hostnames

Next, we found a set of 20 apps that accepted certificates irrespective of the subject name, i. e., if the app wants to connect to `https://www.paypal.com`, it would also accept a certificate issued to `some-domain.com`. We used a certificate for an unrelated domain signed by startSSL[14] for our attacks in this category. The apps leaked information such as credentials for different services, emails, text messages, contact data, bitcoin-miner API keys, premium content or access to online meetings. A particularly interesting finding was an anti-virus app that updated its virus signatures file via a broken SSL connection. Since it seems that the connection is considered secure, no further validation of the signature files is executed by the app. Thus, we were able to feed our own signature file to the anti-virus engine. First, we sent an empty signature database that was accepted, effectively turning off the anti-virus protection without informing the user. In a second attack, we created a virus signature for the anti-virus app itself and sent it to the phone. This signature was accepted by the app, which then recognized itself as a virus and recommended to delete itself, which it also did. Figure 1 shows a screenshot of the result of this attack. This is a very stark reminder that defense in depth is an important security principle. Since the SSL connection was deemed secure, no further checks were performed to determine whether the signature files were legitimate. The app has an install base of 500,000 to one million users.[15]

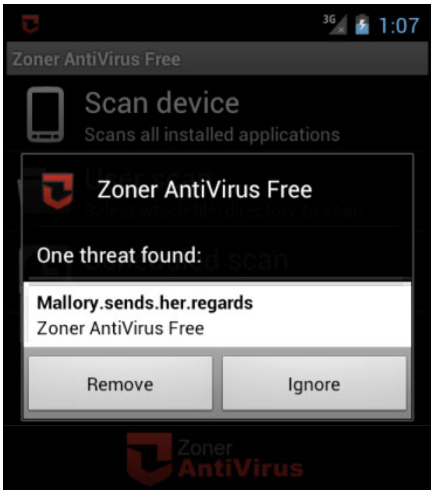

Figure 1: After injecting a virus signature database via a MITM attack over broken SSL, the AntiVirus app recognized itself as a virus and recommended to delete the detected malware.

[14] `https://www.startssl.com/`

[15] honored as the *"Best free anti-virus program for Android" with a detection rate* > 90% – *`http://www.av-test.org/en/tests/android/`*

A second example in this category is an app that offers *"Simple and Secure"* cloud-based data sharing. According to the website, the app is used by 82% of the FORTUNE 500 companies to share documents. It has an install base between 1 and 5 million users. While the app offers simple sharing, it leaks the login credentials during the MITMA. One interesting finding in this app was that the login credentials were leaked from a broken SSL channel while up- and downloads of files were properly secured. However, using the login credentials obtained from the broken channel is sufficient to hijack an account and access the data anyway.

A third example is a client app for a popular Web 2.0 site with an install base of 500,000 to 1 million users. When using a Facebook or Google account for login, the app initiates OAuth login sequences and leaks Facebook or Google login credentials.

We also successfully attacked a very popular cross-platform messaging service. While the app has been criticized for sending messages as plaintext and therefore enabling Eve to eavesdrop, the SSL protection that was intended to secure 'sensitive' information such as registration credentials and the user's contact does not protect from Mallory. For instance, we were able to obtain all telephone numbers from a user's address book using a MITMA. At the time of writing, the app had an install base of 10 to 50 million users.

5.4 SSL Stripping

SSL stripping (cf. Section 2.3) can occur if a browsing session begins using HTTP and switches to HTTPS via a link or a redirect. This is commonly used to go to a secure login page from an insecure landing page. The technique is mainly an issue for Android browser apps, but it can also affect other apps using Android's `webkit.WebView` that do not start a browsing session with a HTTPS site. We found the `webkit.WebView` in 11,038 apps. Two noteworthy examples vulnerable to this attack are a social networking app and an online services client app. Both apps use the webkit view to enhance either the social networking experience or use online services (search, mail, etc.) and have 1.5 to 6 million installs. The two apps start the connection with a HTTP landing page, and we could rewrite the HTTPS redirects to HTTP and thus catch the login credentials for Facebook, Yahoo and Google.

One way to overcome this kind of vulnerability is to force the use of HTTPS, as proposed by the HTTP Strict Transport Security IETF Draft[16], or using a tool such as HTTPS-Everywhere.[17] However, these options currently do not exist for Android. Android's default browser as well as available alternatives such as Chrome, Firefox, Opera or the Dolphin Browser do not provide HTTPS-Everywhere-like features out of the box, nor could we find any add-ons for such a feature.

5.5 Lazy SSL Use

Although the Android SDK does not support SSL pinning out of the box, Android apps can also take advantage of the fact that they can customize the way SSL validation is implemented. Unlike general purpose web browsers that need to be able to connect to any number of sites as ordained by the user, many Android apps focus on a limited number of hosts picked by the app developer: for example, the PayPal app's main interaction is with paypal.com and its sister sites. In such a case, it would be feasible to implement SSL pinning, either selecting the small number of CAs actually used to sign the sites or even pin the precise certificates. This prevents rogue or compromised CAs from mounting MITM attacks against the app. To implement SSL pinning, an app can use its own `KeyStore` of trusted root CA certificates or implement a `TrustManager` that only trusts specific public key fingerprints.

To investigate the usage of SSL pinning, we cherry-picked 20 high profile apps that were not prone to the previous MITM attacks and manually audited them. We installed our own root CA certificate on the phone and set up an SSL MITM proxy that automatically created CA-signed certificates for the hosts an app connects to. Then, we executed MITM attacks against the apps. Table 3 shows the results. Only 2 of the apps make use of SSL pinning and thus were safe from our attack. All other apps trust all root CA signatures, as long as they are part of Android's trust anchors, and thus were vulnerable to the executed attack.

Table 3: Results of the SSL pinning analysis.

App	Installs	SSL Pinning
Amazon MP3	10-50 million	
Chrome	0.5-1 million	
Dolphin Browser HD	10-50 million	
Dropbox	10-50 million	
Ebay	10-50 million	
Expedia Bookings	0.5-1 million	
Facebook Messenger	10-50 million	
Facebook	100-500 million	
Foursquare	5-10 million	
GMail	100-500 million	
Google Play Market	All Phones	
Google+	10-50 million	
Hotmail	5-10 million	
Instagram	5-10 million	
OfficeSuite Pro 6	1-5 million	
PayPal	1-5 million	
Twitter	50-100 million	✓
Voxer Walkie Talkie	10-50 million	✓
Yahoo! Messenger	10-50 million	
Yahoo! Mail	10-50 million	

5.6 Missing Feedback

When an app accesses the Internet and sends or receives data, the Android OS does not provide any visual feedback to the user whether or not the underlying communication channel is secure. The apps are also not required to signal this themselves and there is nothing stopping an app from displaying wrong, misguided or simply no information. We found several apps that provided SSL options in their settings or displayed visual security indicators but failed to establish secure SSL channels for different reasons.

We found banking apps in this category that we could not fully test, since we did not have access to the required bank accounts. However, these apps stated that they were using SSL-secured connections and displayed green visual security indicators, but suffered from one of the MITMA vulnerabilities shown above. We were therefore able to intercept login

[16] `http://android-ssl.org/s/10`
[17] `https://www.eff.org/https-everywhere`

credentials, which would enable us to disable banking cards and gather account information using the app.

We found several prominent mail apps that had issues with missing feedback. Both were dedicated apps for specific online services. The first app with an install base between 10 and 50 million users handled registration and login via a secure SSL connection, but the default settings for sending and receiving email are set to HTTP. They can be changed by the user, but the user needs to stumble upon this possibility first. Meanwhile, there was no indication that the emails were not protected.

An instant messaging app with an install base of 100,000 to 500,000 users transfers login credentials via a non-SSL protected channel. Although the user's password is transferred in encrypted form, it does not vary between different logins, so Eve can record the password and could use it in a replay attack to hijack the user's account.

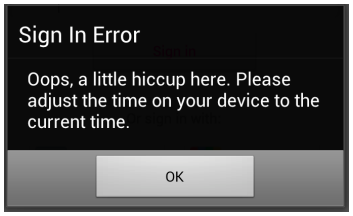

Figure 2: A sample warning message that occurs in an app that is MITM attacked.

We found a framework that provides a graphical app builder, allowing users to easily create apps for Android and other mobile platforms. Apps created with this framework can load code from remote servers by using the `dalvik.system.DexClassLoader`. Downloading remote code is handled via plain HTTP. We analyzed one app built with the framework and could inject and execute arbitrary Java code, since the downloaded code is not verified before execution.

During manual analysis, we also found that 53 apps that were not vulnerable to our MITM attacks did not display a meaningful warning messages to the user under attack. These apps simply refused to work and mostly stated that there were technical or connectivity problems and advised the user to try to reconnect later. There was also an app that recommended an app-update to eliminate the network connection errors. Some apps simply crashed without any announcement. Figure 2 shows a confusing sample error message displayed during a MITMA.

Figure 3: Facebook's SSL warning.

An additional 6 apps not vulnerable to our MITM attacks did display certificate related warning messages, but did not indicate the potential presence of a MITMA. The official Facebook app is not vulnerable to the MITM attacks described above and is a positive example for displaying a meaningful warning message. Even if the warning message contains tech-savvy wording, the user at least has the chance to realize that a MITM attack might be occuring (cf. Fig. 3).

Interestingly – apart from browser apps – there was only one app that allows the user to choose to continue in the presence of an SSL error.

6. LIMITATIONS OF OUR ANALYSIS

This study has the following limitations: a) During static code analysis, the studied applications were selected with a bias towards popular apps; b) The provided install base numbers are only approximate values as provided by Google's Play Market; c) We only checked 100 of the apps where MalloDroid found occurrences of broken SSL implementations manually. For the rest, the existence of the unsafe code does not mean that these apps must be vulnerable to a MITM attack; d) Static code analysis might have failed in some apps, for instance if they were obfuscated. Hence, there might be further vulnerable apps that we did not classify as such; e) During manual audits, the applications were selected with a bias towards popularity and assumed sensitivity of data they handle; f) We could not test the entire workflow of all apps, e. g., it was not possible to create a foreign bank account to see what happens after successfully logging into the bank account.

7. TROUBLE IN PARADISE

The default Android browser is exemplary in its SSL use and uses sensible trust managers and host name verifiers. Also, unlike most special purpose apps, it displays a meaningful error message when faced with an incorrect certificate and allows the user to continue on to the site if (s)he wants to. Thus, it relies on the ability of the user to understand what the displayed warning messages mean and what the safest behavior is. There have been many studies of this issue conducted in the context of desktop browsing. Here, to the best of our knowledge, we present the first survey to investigate the users' perceptions when using secure connections in the Android browser.

7.1 Online Survey

The goal of our online survey was to explore whether or not the user can assess the security of a connection in the Android browser. We wanted to test that a) a user can distinguish a HTTPS connection from a regular HTTP connection and b) how the user perceives an SSL warning message. Previous work has addressed the effectiveness of warning dialogues in several scenarios, mostly for phishing on regular computers (e. g., [5, 24]). Recently, Porter Felt et al. [17] conducted a survey on the prompts informing users of the requested permissions of Android apps during installation. The online survey in this paper is based on a similar design, but studies SSL certificate warnings and visual security indicators in Android's default browser.

Participants were recruited through mailing lists of several universities, companies and government agencies. The study invitation offered a chance to win a 600$ voucher from Amazon for participation in an online survey about Android

smartphone usage. The survey could only be accessed directly from an Android phone. We served the survey via HTTPS for one half of the participants and via HTTP for the other. After accessing a landing page, we showed the participants a typical Android certificate warning message, mimicking the behavior of the Android browser. Subsequently, we asked whether the participants had seen this warning before, if they had completely read its text and how much risk they felt they are warned against. We also wanted to know whether or not they believed to be using a secure connection and their reasons for this belief. Finally, we collected demographic information on technical experience, Android usage, previous experience with compromised credentials or accounts as well as age, gender and occupation. More online survey related information can be found in Appendix A.

7.2 Results

754 participants completed the survey. The average age was 24 years ($sd = 4.01$), 88.3% were students while the rest mainly were employees. 61.9% of our participants did not have an IT-related education or job (non-IT experts in the following) and 23.2% had previous experience with compromised credentials or accounts. Overall, the self-reported technical confidence was high: participants stated a mean value of 4.36 for IT experts and 3.58 for non-experts on a scale from 1 (often asking for help) to 5 (often providing help to others). 51.9% of IT experts and 32.8% of non-IT experts have been using an Android smartphone for more than a year and 57.1% of experts and 69.8% of non-experts had only 25 apps or less installed.

Concerning connection security, we found that 47.5% of non-IT experts believed to be using a secure connection, while the survey was served over HTTP. On top of that, even 34.7% of participants with prior IT education thought that they were using a secure channel when they were not. In both groups, 22.4% were unsure about the protection of their connection. Only 58.9% of experts and 44.3% of non-experts correctly identified that they were using a secure or insecure connection when prompted. The majority of users referred to the URL prefix as the reason for their beliefs and 66.5% of participants that were unsure said that they did not know how to judge the connection security. Those users that were wrongly assuming a secure connection stated that they use a trustworthy provider (47.7%), trust their phone (22.7%) or thought that the address was beginning with https:// even though it was not (21.6%) as a justification for their beliefs. Interestingly, participants that stated that they had suffered from compromised credentials or online accounts before did significantly better in judging the connection state ($\chi^2 = 85.36, df = 6, p < 0.05$).

Concerning the warning message, the majority of participants stated that they had not seen such a certificate warning before (57.6% of non-IT experts and 52.3% of IT experts) or were unsure (5.9%/9.2%). 24.0% of all participants only read the warning partially and 4.5% did not read it at all. These numbers did not differ significantly based on whether or not they had seen the warning before. The participants rated the risk they were warned against with 2.86 ($sd = .94$), with 1 being a very low risk and 5 a very high risk. The perceived risk did not differ significantly between IT-experts and other users.

Overall, the results of our online survey show that assessing the security of a browser session on Android's default browser was problematic for a large number of our participants. While certificate handling is done correctly by the browser app and basic visual security indicators are offered, the user's awareness for whether or not her or his data is effectively protected is frequently incomplete.

7.3 Limitations

Our survey is limited in the following ways: We used official mailing lists to distribute the invitation for the survey. While, on a technical level, this should not affect the trustworthiness of the mail or the survey site - we did not digitally sign the emails and we served the survey with a URL that was not obviously linked to the university. Therefore, the emails could have been spoofed. Nonetheless, it is likely that a higher level of trust was induced in most participants, due to the fact that the survey was advertised as a university study (c.f. [22]). We therefore refrained from evaluating the users' reasons for accepting or rejecting a certificate in this concrete scenario.

Participants were self-recruited from multiple sources, but we received mainly entries from university students for this first exploration. While a study by Sotirakopoulos et al. [23] found little differences between groups of students and the broader population in the usable security context, a more varied sample of participants would improve the general applicability of the results.

8. COUNTERMEASURES

There are several ways to minimize the problem of unencrypted traffic or SSL misuse. They can be categorized into three groups: (1) solutions that are integrated into the Android OS, (2) solutions that are integrated into app markets and (3) standalone solutions.

8.1 OS Solutions

Enforced Certificate Checking.

A radical solution to prevent overly permissive `TrustManager`s, `SSLSocketFactory`s and `AllowAllHostnameVerifier`s is to disallow custom SSL handling completely. This can be achieved by forcing developers to use the standard library implementations provided by Android's APIs. By limiting the way `TrustManager`s, `SSLSocketFactory`s and `HostnameVerifier`s can be used, most cases of faulty code and unintended security flaws could be avoided.

HTTPS Everywhere.

A solution to improve a fair number of the vulnerabilities discovered in our sample set would be an Android version of HTTPS-Everywhere, integrated into the communication APIs. This would prevent most SSL stripping attacks we found in our sample set.

Improved Permissions and Policies.

Instead of simply having a general permission for Internet access, a more fine-grained policy model could allow for more control (cf. [16]). By introducing separate permissions for `INTERNET_SSL` and `INTERNET_PLAIN`, apps could indicate which type of connections is used. This would give users a chance to avoid applications that do not use HTTPS at all. However, in mixed-mode cases or when SSL is used

but used incorrectly, this method would not protect the user without additional indicators/countermeasures. Furthermore, introducing policies like `GSM_ONLY`, `NO_OPEN_ WIFI` or `TRUSTED_NETWORKS` could help to protect apps from some MITM attacks. Despite the fact that cellular networks such as GSM/3G/4G do not provide absolute security, they still require considerably more effort to execute an active MITMA. Apps could then specify which types of networks or even which connections specifically are allowed to be used. However, this countermeasure could have considerable usability and acceptance issues.

Visual Security Feedback.

Reasonable feedback to the user about the security status of the currently running application is undoubtedly a valuable countermeasure – at least for some users. The operating system should provide visual feedback on whether or not apps are communicating via a secure channel. Current mobile devices usually only show the signal strength, the connection type and whether any transfers are in progress at all. Finding an effective way to inform users about which apps are currently communicating with the Internet and whether the communication is secure is not trivial and should be studied carefully before a solution is propagated.

MalloDroid Installation Protection.

MalloDroid could be integrated into app installers, such as Kirin [9], to perform static code analysis at install time. This analysis performed directly on a phone could warn of potentially unsafe applications. Users would then have to decide whether they wish to install the app irrespective of the warning.

8.2 App Market Solutions

Similar to the MalloDroid installation protection, MalloDroid could be integrated into app markets. This form of automated checking of apps could either be used to reject apps from entering the market or warnings could be added to the app's description. Both options have usability and acceptance issues that need to be studied.

8.3 Standalone Solution: The MalloDroid App & Service

All countermeasures mentioned above require modification of the Android OS and support from vendors and/or app markets. Standalone solutions can be deployed more easily. Therefore, as a stop-gap measure, we are going to offer our MalloDroid tool as a Web app. This will at least allow interested users to perform checks on apps before they install them. MalloDroid can of course also be used as-is with Androguard.

9. CONCLUSION

In this paper, we presented an investigation of the current state of SSL/TLS usage in Android and the security threats posed by benign Android apps that communicate over the Internet using SSL/TLS. We have built MalloDroid, a tool that uses static code analysis to detect apps that potentially use SSL/TLS inadequately or incorrectly and thus are potentially vulnerable to MITM attacks. Our analysis of the 13,500 most popular free apps from the Google Play Market has shown that 1,074 apps contain code belonging to this category. These 1,074 apps represent 17.0% of the apps that contain HTTPS URLs. To evaluate the real threat of such potential vulnerabilities, we have manually mounted MITM attacks against 100 selected apps from that set. This manual audit has revealed widespread and serious vulnerabilities. We have captured credentials for American Express, Diners Club, Paypal, Facebook, Twitter, Google, Yahoo, Microsoft Live ID, Box, WordPress, IBM Sametime, remote servers, bank accounts and email accounts. We have successfully manipulated virus signatures downloaded via the automatic update functionality of an anti-virus app to neutralize the protection or even to remove arbitrary apps, including the anti-virus program itself. It was possible to remotely inject and execute code in an app created by a vulnerable app-building framework. The cumulative number of installs of apps with confirmed vulnerabilities against MITM attacks is between 39.5 and 185 million users, according to Google's Play Market.

The results of our online survey with 754 participants showed that there is some confusion among Android users as to which security indicators are indicative of a secure connection, and about half of the participants could not judge the security state of a browser session correctly. We discussed possible countermeasures that could alleviate the problems of unencrypted traffic and SSL misuse. We offer MalloDroid as a first countermeasure to possibly identify potentially vulnerable apps.

The findings of our investigation suggest several areas of future work. We intend to provide a MalloDroid Web App and will make it available to Android users. Moreover, there seems to be a need for more education and simpler tools to enable easy and secure development of Android apps. But most importantly, research is needed to study which countermeasures offer the right combination of usability for developers and users, security benefits and economic incentives to be deployed on a large scale.

10. ACKNOWLEDGEMENT

The authors would like to thank Marten Oltrogge and Felix Fischer for their help during app analysis and the anonymous reviewers for their helpful comments.

11. REFERENCES

[1] X.509 Internet Public Key Infrastructure, Online Certificate Status Protocol - OCSP. `http://tools.ietf.org/html/rfc2560`.

[2] RFC 5280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. `http://tools.ietf.org/html/rfc5280`, 2008.

[3] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A. Sadeghi, and B. Shastry. Towards Taming Privilege-Escalation Attacks on Android. In *Proceedings of the 19th Network and Distributed System Security Symposium*, 2012.

[4] L. Davi, A. Dmitrienko, A. Sadeghi, and M. Winandy. Privilege Escalation Attacks on Android. In *Proceedings of the 13th International Conference on Information Security*, pages 346–360, 2011.

[5] S. Egelman, L. Cranor, and J. Hong. You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings. In *Proceedings of the*

26th Annual SIGCHI Conference on Human Factors in Computing Systems, pages 1065–1074, 2008.

[6] A. Egners, B. Marschollek, and U. Meyer. Messing with Android's Permission Model. In *Proceedings of the IEEE TrustCom*, pages 1–22, 2012.

[7] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth. TaintDroid: An Information-flow Tracking System For Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, pages 393–407, 2010.

[8] W. Enck, D. Octeau, P. McDaniel, and S. Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX Conference on Security*, 2011.

[9] W. Enck, M. Ongtang, and P. McDaniel. On Lightweight Mobile Phone Application Certification. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 235–245, 2009.

[10] W. Enck, M. Ongtang, and P. McDaniel. Understanding Android Security. In *Proceedings of the IEEE International Conference on Security & Privacy*, pages 50–57, 2009.

[11] C. Jackson and A. Barth. ForceHTTPS: Protecting High-security Web Sites From Network Attacks. In *Proceeding of the 17th International Conference on World Wide Web*, pages 525–534, 2008.

[12] M. Marlinspike. More Tricks For Defeating SSL In Practice. In *Black Hat USA*, 2009.

[13] M. Marlinspike. New Tricks for Defeating SSL in Practice. In *Black Hat Europe*, 2009.

[14] P. McDaniel and W. Enck. Not So Great Expectations: Why Application Markets Haven't Failed Security. *IEEE Security & Privacy*, 8(5):76–78, 2010.

[15] M. Nauman, S. Khan, and X. Zhang. Apex: Extending Android Permission Model And Enforcement With User-defined Runtime Constraints. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 328–332, 2010.

[16] A. Porter Felt, E. Chin, S. Hanna, D. Song, and D. Wagner. Android Permissions Demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, pages 627–638, 2011.

[17] A. Porter Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner. Android Permissions: User Attention, Comprehension, and Behavior. In *Proceedings of the 8th Symposium on Usable Privacy and Security*, 2012.

[18] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile Protection for Smartphones. In *Proceedings of the 26th Annual Computer Security Applications Conference*, pages 347–356, 2010.

[19] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, and C. Glezer. Google Android: A Comprehensive Security Assessment. *Security & Privacy, IEEE*, 8(2):35–44, 2010.

[20] D. Shin and R. Lopes. An Empirical Study of Visual Security Cues to Prevent The SSLstripping Attack. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 287–296, 2011.

[21] Y. Song, C. Yang, and G. Gu. Who is Peeping at Your Passwords at Starbucks? – To Catch An Evil Twin Access Point. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 323–332, 2010.

[22] A. Sotirakopoulos and K. Hawkey. "I Did it Because I Trusted You": Challenges With The Study Environment Biasing Participant Behaviours. In *Proceedings of the 6th Symposium on Usable Privacy and Security*, 2010.

[23] A. Sotirakopoulos, K. Hawkey, and K. Beznosov. On the Challenges in Usable Security Lab Studies: Lessons Learned From Replicating a Study on SSL Warnings. In *Proceedings of the 7th Symposium on Usable Privacy and Security*, 2011.

[24] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. Cranor. Crying Wolf: An Empirical Study of SSL Warning Effectiveness. In *Proceedings of the 18th USENIX Security Symposium*, pages 399–416, 2009.

[25] T. Vidas, D. Votipka, and N. Christin. All Your Droid Are Belong To Us: A Survey Of Current Android Attacks. In *Proceedings of the 5th USENIX Workshop on Offensive Technologies*, pages 10–10, 2011.

[26] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. In *Proceedings of the 19th Annual Network and Distributed System Security Symposium*, 2012.

APPENDIX

A. ONLINE SURVEY

We based the questions of our online survey on previous surveys [23], [22] and [17], adapting them to our scenario and optimizing the survey for mobile delivery. For this purpose, we removed most of the free text answers and replaced them by multiple choice or radio button answers to make the online survey easier to handle on an Android smartphone.

As described in Section 7.1, after clicking a link on the landing page to begin the study, participants were redirected to a non-university domain with a page designed to look like Android's 4.0 default browser warning message. The warning message was interactive, hence users could click on "Certificate Details" for more information. The page thus replicated the user experience of a real SSL warning message in Android's default browser.

We presented two different SSL warnings, although, just as with the real Android SSL warnings, the difference only became visible if the user clicked on "Certificate Details". One warning stated that the certificate was signed by an untrusted CA and the other warning stated that the hostname did not match the certificate's common name.

We tracked whether the participants clicked "Continue" or "Cancel". In both cases, participants were directed to the first page of the questionnaire that explained that the message just shown was part of the study. For half of the participants, the study was served via HTTPS, and for the other half, it was served via plain HTTP. Hence, we had four different groups: `untrustedCA+HTTP`, `untrustedCA+HTTPS`, `wronghostname+HTTP` and `wronghostname+HTTPS`. The survey was also hosted on a domain that did not obviously belong to our universities, in order to avoid the implicit

trust often associated with university servers. Unlike previous studies ([23], [22] and [17]), we did not refer to the SSL warning message as a warning message during the online survey. Instead, we called it a popup message to use a neutral term avoiding a bias in the users' perceptions. Subsequently, questions contained in the online survey are listed. In addition to SSL warning message comprehension, HTTPS indicator comprehension, Android usage and online security awareness, we asked the participants about their self-reported technical expertise and demographic information. Due to space constraints, questions from the last two categories are not listed below.

A.1 SSL Warning Message Comprehension

- The popup message you just saw is part of this survey. Have you previously seen this kind of message while surfing the Internet with your Android phone?
  - (Yes, No, I'm not sure)
- Did you read the entire text of the popup message?
  - (Yes, Only partially, No)
- Please rate the following statements (all statements were rated on a 5-point Likert scale, ranging from "Don't agree" to "Totally agree"):
  - I always read these kind of popup messages entirely.
  - I understood the popup message.
  - I am not interested in such popup messages.
  - I already knew this popup message.
  - I am only interested in winning the voucher.
- When you saw the popup message, what was your first reaction?
  - I was thankful for the message.
  - I was annoyed by the popup.
  - I didn't care.
  - Other: (text field)
- Please rate the amount of risk you feel you were warned against.
  - 5-point Likert scale ranging from "Very low risk" to "Very high risk"
- What action, if any, did the popup message want you to take?
  - To not continue to the website.
  - To be careful while continuing to the website.
  - To continue to the website.
  - I did not feel it wanted me to take any action.
  - Other: (text field)
- How much did the following factors influence your decision to heed or ignore the popup message? (all factors were rated on a 5-point Likert scale, ranging from "Very little influence" to "Very high influence")
  - The text of the message.
  - The colors of the message.
  - The choices that the message presented.
  - The destination URL.
  - The chance to win a voucher.
  - The fact that this is an online survey.
  - Other factors: (text field)
- Which factor had the most influence on your decision?
  - The text of the message.
  - The colors of the message.
  - The choices that the message presented.
  - The destination URL.
  - The chance to win a voucher.
  - The fact that this is an online survey.
  - Other factors: (text field)

A.2 HTTPS Indicator Comprehension

- Is the Internet connection to this online survey secure?
  - (Yes, No, I'm not sure)
- Please explain your decision:
  if answered with "yes"
  - I trust my service provider.
  - I trust my smartphone.
  - The URL starts with `https://`.
  - All Internet communication is secure.
  - A lock symbol is visible in the browser bar.
  - Other: (text field)

  if answered "no"
  - I do not trust my service provider.
  - I do not trust my smartphone.
  - The URL starts with `http://`
  - Communicating over the Internet is always insecure.
  - There is no lock symbol in the browser bar.
  - The address bar is not green.
  - Other: (text field)

  if answered "don't know"
  - I don't know how to determine this.
  - I don't care.
  - I don't trust the visual indicators.
  - I don't trust IT in general.
  - Other: (text field)

A.3 Android Usage

- For how long have you been using an Android smartphone?
  - 1 month or less
  - 2 - 6 months
  - 7 - 11 months
  - 1 - 2 years
  - more than 2 years
- Did you turn off browser warning messages?
- How many apps have you installed on your phone?

A.4 Online Security Awareness

- Have you ever had any online account information stolen? (Yes, No)
- Have you ever found fraudulent transactions on a bank statement? (Yes, No)
- Have you ever been notified that your personal information has been stolen or compromised? (Yes, No)
- Have you ever lost your smartphone? (Yes, No)