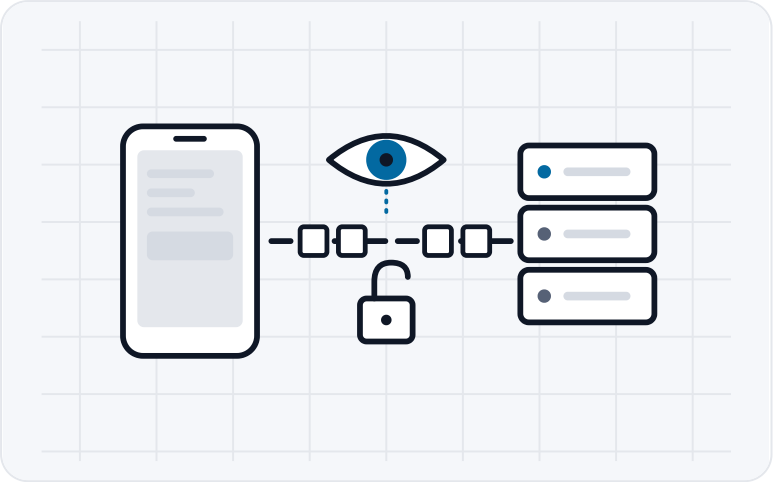
A-SIT Technology
E-Government Innovation Center

26 January 2023 Analysis · Mobile Security · Web & Cloud · 17 min read

Prevalence of HTTP in Android applications

Florian Draschbacher



Authors Florian Draschbacher
Version 2023-01-25
Source https://technology2.a-sit.at/en/publications/prevalence-of-http-in-android-applications/
Note Machine translation.

Key findings

- 163 of the 1,604 runnable apps (10%) make unprotected HTTP connections to 144 different hosts.
- Of the 144 hosts contacted over HTTP, 128 were reachable; 109 also ran an HTTPS server, and 84 of those had a valid TLS certificate.
- 44.9% of apps with their own Network Security Configuration allow HTTP app-wide; domain-specific entries allow it in 88% of cases.
- The host most often contacted over HTTP is ip-api.com (14 apps); the next two belong to the ad mediation provider AdMost.
- The share of apps using HTTP fell from almost a third (Piccolboni et al., 2020) to around 10%; ad providers remain a cause of insecure NSCs.

In contrast to the web ecosystem, where after initial problems the encryption of HTTP connections through TLS has gained wide adoption, a considerable amount of Android applications still employ cleartext HTTP connections. This is particularly problematic in applications that transmit sensitive user information (e.g. account data) to a server. In this scenario, a man-in-the-middle attacker can almost effortlessly stealthily extract information, e.g. through an ARP spoofing or Evil Twin attack.

As part of this research project, we shine light on the general problem and describe in particular the situation of apps for the mobile operating system Android. To this end, we provide an overview of relevant scientific studies from the past before we introduce and discuss results from our own analysis on a data set of recent popular Android applications.

1. Introduction

When the World Wide Web was conceived at the beginning of the 1990s, the use cases envisaged were primarily about consuming information. Data was therefore meant to flow mainly from a central storage point (server) to the consumer (client, often a web browser) on the end user's device. Conceptually, the design thus followed that of libraries, which had been the most established form of information storage for centuries. However, it soon became apparent that the new technology was also very well suited to realising entirely new paradigms in which the information delivered by the server was tailored to the individual user. To this end, personal data was now sometimes sent from the user to the server as part of the communication. It soon became clear that some use cases, such as the processing of financial transactions, required the exchange of information that must not fall into the hands of third parties at any price.

Gradually, an awareness emerged that the HTTP protocol offered no protection against third parties between browser and server. Even if no sensitive data is transmitted from the user to the browser, the mere information that a particular page has been accessed can be sensitive and should be shielded from the eyes of observers in the interest of privacy. Moreover, it was not only trivial for attackers to read the data exchange and thereby obtain information about the user, but also to inject false data in a targeted manner.

To remedy these considerable security problems, a secured version of the HTTP protocol for communication between browser and server was eventually developed. The new protocol, called HTTPS, has an additional security layer, which was named Secure Socket Layer (SSL) and later Transport Layer Security (TLS). Based on cryptography, this additional layer makes it possible to establish the identity of a server unambiguously and to protect the communication against inspection or manipulation by third parties.

To benefit from the security guarantees of the HTTPS protocol, however, server and client had to be adapted accordingly. Only if both server and client support HTTPS, and the client initiates the connection setup via this protocol, is the further communication over the connection actually protected.

After the availability and use of HTTPS rose only slowly during the first years following its development, the necessary security awareness was eventually created among a substantial share of browser users. This was achieved, among other things, by clearly highlighting secure connections in the user interface of the common web browsers and by targeted warnings for insecure connections.

While the originally addressed problem could thus gradually be largely solved in traditional server-browser configurations, the introduction of smartphones had in the meantime given rise to an additional problem. Smartphones had increasingly established themselves as an always-available means of accessing the internet. In contrast to classic use cases, however, the browser was often no longer the interface between user and server. Instead, application-specific client applications enabled a more seamless integration of web services into mobile operating systems.

Because these client applications are developed and offered by the respective service providers or website operators, it is considerably more difficult to bring about a comprehensive upgrade to HTTPS here. Users have no way of determining whether an app communicates with its backend server over a secured connection or not.

By summarising scientific publications and drawing on our own investigations, this report aims to shed light on how high the share of apps with insecure connections is among current Android applications.

2. Background

This section explains the technologies that are necessary for understanding the later discussion.

2.1. HTTP

The HyperText Transfer Protocol was developed by Tim Berners-Lee in 1990 as part of the World Wide Web and is today one of the central pillars of the modern internet. It serves the communication between a client, which usually requests data, and a server, which processes and answers requests. The subject of the transmission is mostly HTML content, which is displayed by web browsers. In the case of application-specific client applications, the communication often takes place with a web server that provides a REST interface via HTTP. In the broadest sense, this is a way of accessing a database, which is often equipped with authentication, so that the exact extent of data access can be controlled individually for each user.

From a technical point of view, the HTTP protocol is a protocol at the application layer of the protocol stack. Below HTTP is the TCP protocol, which in turn builds on the IP protocol.

2.2. TLS/SSL

TLS is a cryptographic protocol used in the HTTPS stack to ensure the confidentiality, integrity and authenticity of a communication connection between a server and a client. Confidentiality here means that the content of the communication cannot be decrypted by third parties. Integrity means that a message cannot be manipulated by third parties on its way between the two communication participants. In the case of HTTPS, authenticity usually means that the client can be sure of the server's identity. It is thus not possible for an attacker to assume the identity of the server and thereby take over communication intended for the legitimate server.

2.3. HTTPS

Contrary to what the name implies, HTTPS is, from a technical point of view, not a self-contained secure alternative to the HTTP protocol. Rather, HTTPS denotes the use of the existing HTTP protocol over a secured layer of the Transport Layer Security protocol.

To benefit from the security properties of HTTPS and the TLS layer, not only must the server support the protocol, but the client must also use a corresponding HTTPS URL to contact the server. In the case of a web browser as the client, this means that the user must enter a corresponding HTTPS URL in the address bar of the program or follow a link to such a URL.

From the presentation of the first public SSL specification in 1999 [1], it took several years until HTTPS came into widespread use. As late as 2017, estimates for the prevalence of HTTPS support among web servers were around 50% [2]. Measurements in popular web browsers at the same time showed that about half of all server connections also ran over HTTPS. While server-side support for HTTPS has improved massively since then, many connections still ran over less secure HTTP, because many old links still pointed to HTTP URLs and no redirect to HTTPS had been implemented [3]. To get this problem under control, various solutions were presented with which web browsers automatically upgrade URLs on the client side when the server supports HTTPS.

If the client program is an application-specific client application, the HTTPS URL must be provided accordingly by the app developer. A comprehensive solution for automatic upgrading is generally not possible here owing to the diversity of client implementations.

3. Situation on Android

3.1. Background

Android, an open-source software project by Google, is the most widely used operating system for mobile devices such as smartphones and tablets. Owing to their portability, these products are used particularly often for mobile internet access. In many cases, web servers are accessed here not via a browser but via a native client application, which can be integrated particularly closely into the rest of the operating system and can thus offer a better user experience than a website.

To get the problem of insecure HTTP connections under control on the Android platform, Google gradually introduced new security features for app developers starting with Android 6.0. The first step was the usesCleartextTraffic flag in the application manifest (where developers tell the system various properties of their app). If the flag was disabled (and the security feature thus enabled), the system prevented the app from using HTTP in its server communication. Owing to the lack of granularity, however, this security feature could only be used if the developer could ensure that the app never has to access a server that does not support HTTPS. This excluded, for example, apps that deliberately used HTTP for server accesses that are not security-relevant. In addition, the flag was enabled by default, so that knowledge and initiative on the part of the developer were required to activate the security feature.

Android 7.0 therefore introduced an XML-based configuration file called Network Security Configuration (NSC), which allows developers to configure, in a straightforward way and application-wide, a precise policy for an app's network connections. This makes it possible to block HTTP in general but allow it for selected domains.

Starting with Android 9.0, all HTTP connections are blocked in the default configuration, so that developers are inevitably confronted with the security feature if they want to use an HTTP connection. To preserve backward compatibility, however, the new default only applies to apps whose developers specify Android 9

or newer as the target API in the application manifest. To achieve a comprehensive effect despite this restriction, Google enforces for apps in the Play Store that a current target API is aimed for.

3.2. Research

As part of the ever broader scientific interest in topics of mobile security, studies devoted to analysing the security of client applications have also emerged in the recent past. They showed that the spread of HTTPS is progressing only very slowly here. In 2018, based on an analysis of version histories of Android apps, Ren et al. found that for a given web domain it took 5 years from server-side support for HTTPS until half of all apps accessing it did so via HTTPS [4]. Across all versions, almost 70% of all applications used HTTP connections for their core functionality. The researchers suspect that developers are too hesitant, for example when updating the library dependencies of their software projects. Piccolboni et al. [5] reported in 2020 that about one third of 1780 automatically examined Android applications established HTTP connections.

Since the introduction of the NSC system, its effects have also been studied. Possemato et al. [6] concluded in 2020 that, despite the introduction of the NSC system and the enforcement of HTTPS via the target API on Google Play, 67% of the examined applications still allowed HTTP communication. The researchers also found that some providers of in-app advertising specifically instruct app developers to allow HTTP globally, because advertisements were in part delivered over insecure connections. In a similar study from 2021, Oltrogge et al. [7] also came to the conclusion that almost 90% of the apps that defined an NSC did so to weaken the security of the default configuration.

4. Investigation

The most recent studies on the prevalence of HTTP connections in Android applications are already several years old. Since the research field of mobile security is particularly fast-moving, it can be assumed that the data collected at the time no longer corresponds to current reality. For this reason, we repeat similar investigations on a data set of current popular Android applications.

4.1. Data set

The Android applications examined here were collected from the Google Play Store in 2022. They are the 100 most popular applications in Austria from each of 33 different categories. Using a self-developed helper program, all APK files for the x86 platform were downloaded for each application. Because some applications were not compatible with this platform, a total of 2697 applications were downloaded.

4.2. Test setup and methodology

Owing to the size of our data set, the first step of the analysis was carried out automatically. For this purpose, we constructed a testbed based on the Android emulator. Each application was run on the emulator and driven by random user input. For this, the emulator was configured so that every network connection ran through a proxy server. This was set up on the basis of the open-source program mitmproxy[1] so that all insecure HTTP connections were saved in a log file. Of the 2697 downloaded applications, 1604 could be run in our testbed.

The analysis was run on a bare-metal-virtualised Ubuntu 20.04 LTS system with 128 CPU cores and 128 GB RAM, which ran on a dual AMD EPYC 7502 CPU with 1 TB RAM. This setup made it possible to run 6 instances of our analysis testbed in parallel. Each instance received not only its own virtual Android device in the emulator but also its own proxy server. This ensured that the network traffic could be attributed to the one running app in each case.

Based on the log file, a second automated analysis of the web servers found was finally carried out. For this purpose, it was determined for each server contacted via HTTP whether an HTTPS connection would also have been possible. If the connection was established successfully, it was checked whether the server also uses a valid TLS certificate signed by a certificate authority.

As the last step of our analysis, we also examined the Network Security Configurations of the apps in our data set. For this purpose, two Python scripts were developed that can extract the corresponding XML file from an APK package and then analyse it.

4.3. Prevalence of HTTP

Our investigation shows that 163 apps in our data set (10%) establish unprotected HTTP connections to 144 different hosts. All hosts that were contacted via HTTP by 3 or more different apps are summarised in Table 1 Most frequently contacted hosts via HTTP.

The most frequently contacted host, ip-api.com, serves the geolocation of users based on their IP address. That HTTPS is not used here is unproblematic from a data protection point of view. Even when HTTPS is used, the IP addresses of the two communication participants cannot be protected, since they are needed for establishing the TLS connection in the first place. Nevertheless, the use of HTTP is problematic here in the Android context, since it forces the developer to relax the default NSC. If the developer is not careful, they may under certain circumstances enable cleartext traffic for the entire application.

Host	Number of apps
ip-api.com	14
cdn-api.admost.com	8
med-api.admost.com	7
link.mobihealthplus.com	7
player.vimeo.com	3
onex-28c2.kxcdn.com	3
4dwallpapers.com	3
examcontrol.com	3
superpowered.com	3

Table 1 Most frequently contacted hosts via HTTP

The next two hosts, measured by frequency of use, belong to AdMost, a provider of advertising mediation. This is an advertising broker whose task is to procure for app providers those advertisements that currently pay best. In the documentation of the AdMost Android SDK[2], the vendor describes in detail how developers must set up an insecure Network Security Configuration in order to use AdMost. In the industry, it is not unusual for user-related data to be transmitted to the advertising provider in order to maximise profit through personalised advertising. That an unsecured connection is used here is worrying.

Vimeo is a video portal that Android applications can access via a dedicated Software Development Kit. Why access takes place via HTTP here is unclear. It can be assumed that app developers are using an outdated version of the SDK here. In the current version, we could not identify any need for HTTP (and corresponding changes to the NSC).

Superpowered is a provider of a software library for audio processing. It can be assumed that the library contacts a server either as a licence check or to collect analytics data. Whether the contact transmits sensitive data was not examined here. Since the connection is established from native code, the NSC is ignored. It therefore does not have to be disabled by the app developer.

All other frequently contacted hosts concern the core functionality of the respective apps and are attributable to their developers. These hosts appear in the ranking because several apps by the same developer access them.

4.4. Availability of HTTPS

Of the 144 different hosts contacted via HTTP, 128 (89%) could also be reached by our automated test program. On 109 of these hosts (85% of the hosts reachable via HTTP), an HTTPS server was also reachable (open server socket on the HTTPS standard port 443). Of these, 84 (77%) also had a valid TLS certificate, so that a connection could be established without problems. Figure 1 illustrates the distribution of the hosts by degree of HTTPS support.

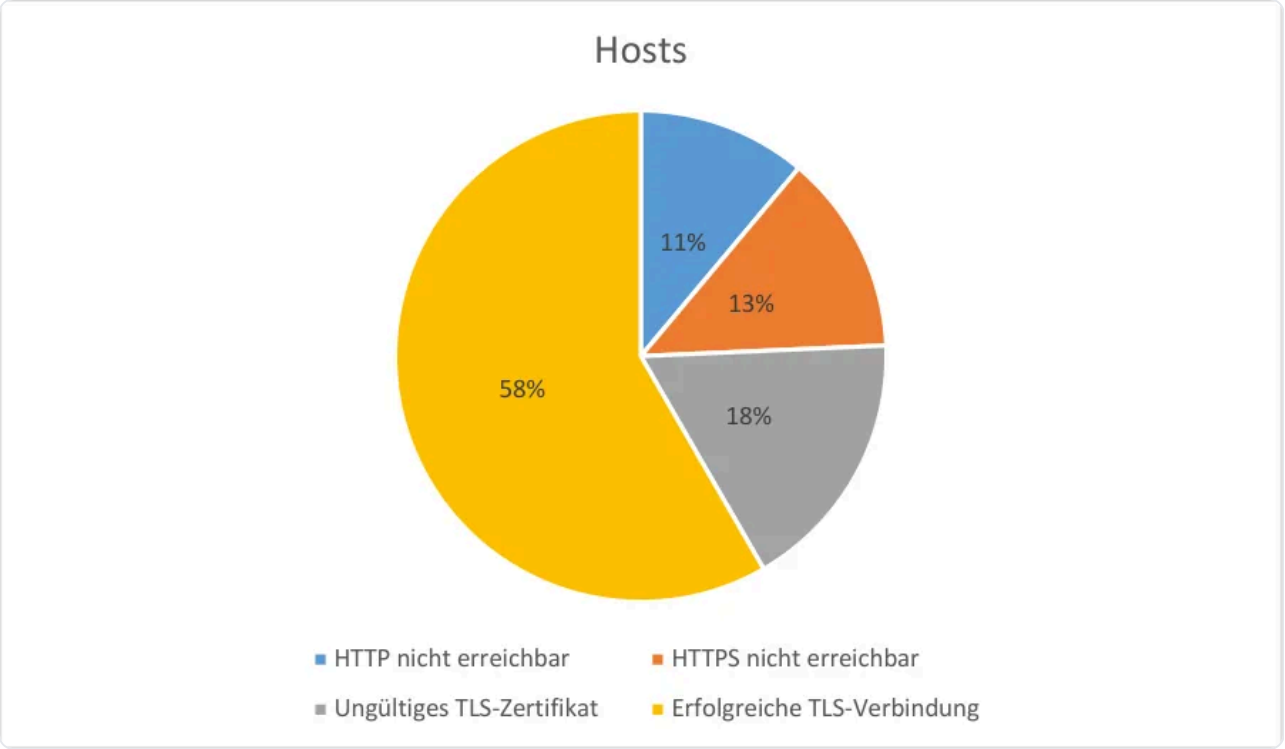


Figure 1: Hosts contacted via HTTP by HTTPS compatibility

4.5. Examination of the Network Security Configurations

40.5% of the 2697 applications in our data set used their own Network Security Configuration. Of the remaining 1603 apps, 1534 (96%) specified a target SDK that prohibits the use of HTTP in the default configuration. 44.9% of the apps with an NSC used a base configuration that allows unencrypted HTTP communication application-wide. Domain-specific configurations, too, were primarily used to allow HTTP communication, namely in 88% of cases. Figure 2 shows this distribution graphically.

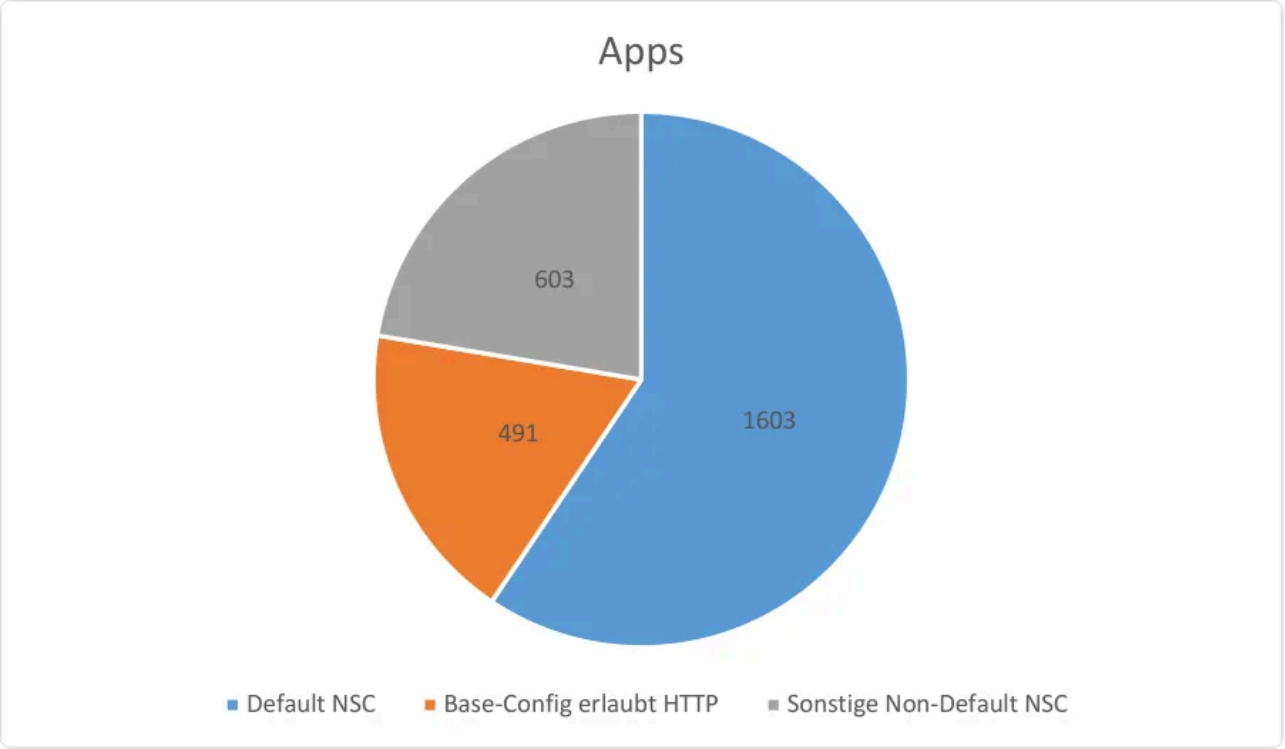


Figure 2: Clustering of the most common NSC properties

4.6. Interpretation of the findings

Although our investigations show that the security of the internet connections of Android apps has improved slightly since the last scientific studies on the subject, there is still much room for improvement. Positive developments can be observed particularly in the number of apps that use HTTP connections. While Piccolboni et al. [5] still found in 2020 that almost a third of all apps were affected, we found the problem in only about 10% of the examined applications. The findings of Oltrogge et al. [7] regarding the Network Security Configuration, on the other hand, we could essentially confirm. Their publication documents 57123 apps that globally allow HTTP communication, among 96400 apps with a non-default NSC. This corresponds to 59.2%. Our investigation on a data set 2 years more recent found a share of a similar order of magnitude of 44.9%. We could also confirm in current apps the finding of Possemato et al. [6] that advertising providers are in part responsible for insecure NSCs.

5. Summary

In this report, we examined the prevalence of HTTP in Android applications. An overview was given of the problem of unprotected HTTP connections, the benefits of TLS and the difficulties of its deployment. In addition, the situation on Android was examined, in particular with regard to HTTP connections in client applications. After summarising relevant literature, we documented the results of our own investigations of a data set of current Android applications. We were able to show that, although a positive trend can be noted, many problems that science has pointed out in the past nevertheless persist.

References

[1] T. Dierks, „The TLS Protocol Version 1.0," January 1999. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc2246. [Zugriff am 25 January 2023].

[2] A. P. Felt, R. Barnes, A. King, C. Palmer, C. Bentzel und P. Tabriz, „Measuring HTTPS Adoption on the Web," in Proceedings of the 26th USENIX Security Symposium, Vancouver, 2017.

[3] C. Kerschbaumer, J. Gaibler, A. Edelstein und T. Van der Merwe, „HTTPS-Only: Upgrading all connections to https in Web Browsers," in Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb) 2021, Virtual, 2021.

[4] J. Ren, M. Lindorfer und D. J. Dubois, „Bug Fixes, Improvements, ... and Privacy Leaks," in Network and Distributed Systems Security (NDSS) Symposium 2018, San Diego, 2018.

[5] L. Piccolboni, G. D. Guglielmo, L. P. Carloni und S. Sethumadhavan, „CRYLOGGER: Detecting Crypto Misuses Dynamically," in IEEE Symposium on Security and Privacy, San Francisco, 2021.

[6] A. Possemato und Y. Fratantonio, „Towards HTTPS Everywhere on Android: We Are Not There Yet," in Proceedings of the 29th USENIX Security Symposium, 2020.

[7] M. Oltrogge, N. Huaman, S. Amft, Y. Acar, M. Backes und S. Fahl, „Why Eve and Mallory Still Love Android: Revisiting TLS (In)Security in Android Applications," in Usenix Security Symposium 2021, Vancouver, 2021.

[8] Y. Zhou, K. Patel, L. Wu, Z. Wang und X. Jiang, „Hybrid User-level Sandboxing of Third-party Android Apps," in Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, 2015.

---

1. https://mitmproxy.org ↩
2. https://admost.github.io/amrandroid/ ↩