On the (In)Security of Mobile Two-Factor Authentication

Alexandra Dmitrienko[2], Christopher Liebchen[1],
Christian Rossow[3], and Ahmad-Reza Sadeghi[1]

[1] CASED/Technische Universität Darmstadt, Germany
[2] CASED/Fraunhofer SIT Darmstadt, Germany
[3] Vrije Universiteit Amsterdam, The Netherlands
email:{christopher.liebchen,ahmad.sadeghi}@trust.cased.de,
alexandra.dmitrienko@sit.fraunhofer.de, c.rossow@vu.nl

Abstract. Two-factor authentication (2FA) schemes aim at strengthening the security of login password-based authentication by deploying secondary authentication tokens. In this context, mobile 2FA schemes require no additional hardware (e.g., a smartcard) to store and handle the secondary authentication token, and hence are considered as a reasonable trade-off between security, usability and costs. They are widely used in online banking and increasingly deployed by Internet service providers.
In this paper, we investigate 2FA implementations of several well-known Internet service providers such as Google, Dropbox, Twitter and Facebook. We identify various weaknesses that allow an attacker to easily bypass them, even when the secondary authentication token is not under attacker's control. We then go a step further and present a more general attack against mobile 2FA schemes. Our attack relies on cross-platform infection that subverts control over both end points (PC and a mobile device) involved in the authentication protocol. We apply this attack in practice and successfully circumvent diverse schemes: SMS-based TAN solutions of four large banks, one instance of a visual TAN scheme, 2FA login verification systems of Google, Dropbox, Twitter and Facebook accounts, and the Google Authenticator app currently used by 32 third-party service providers. Finally, we cluster and analyze hundreds of real-world malicious Android apps that target mobile 2FA schemes and show that banking Trojans already deploy mobile counterparts that steal 2FA credentials like TANs.

Keywords: Two-factor authentication, Smartphones Security, Banking Trojans, Cross-platform Infection

1 Introduction

The security and privacy threats through malware are constantly growing both in quantity and quality. In this context the traditional login/password authentication is considered insufficiently secure for many security-critical applications such as online banking or login to personal accounts. Two-factor authentication (2FA) schemes promise a higher protection level by extending the single authentication factor, i.e., *what the user knows*, with other authentication factors such as *what the user has* (e.g., a hardware token or a smartphone), or *what the user is* (e.g., biometrics) [38].

Even if one device/factor (e.g., PC) is compromised – a typical scenario nowadays – the chance of the malware to gain control over the second device/factor (e.g., mobile device) simultaneously is considered to be very low.

While the biometric-based authentication is relatively expensive and raises privacy concerns, One Time Passwords (OTPs) offer a promising alternative for 2FA systems. For instance, hardware-based tokens such as OTP generators [36] are less costly, but still generate additional expenses for users and are inconvenient, particularly when the user needs to carry additional hardware tokens for different organizations (e.g., for accounts at several banks). On the other hand, 2FA schemes that use mobile devices (such as smartphones) to handle OTPs have become popular recently, and have been adapted by many banks and large service providers. These *mobile 2FA* schemes are considered to provide an appropriate trade-off between security, usability and cost, and will be the focus of this paper.

A prominent example of mobile 2FA are SMS-based TAN systems (known as mTAN, smsTAN, mobileTAN and a like). Their goal is to mitigate account abuse even if the banking login credentials have been compromised, e.g., by a PC-based banking Trojan. Here, the service provider (i.e., the bank) generates a Transaction Authentication Number (TAN), which is a transaction-dependent OTP, and sends it over SMS to the customer's phone. The user/customer needs to confirm a banking transaction by entering this TAN into the other device (typically a PC). Alternatively, visual TAN schemes encrypt and encode the TAN into a 2D barcode (visual cryptogram) which is displayed on the customer's PC from where it is photographed and decrypted by the corresponding app on the smartphone. SMS-based TAN schemes are widely deployed worldwide (USA, UK, China, Europe)[4]. Further, some large European banks have already adapted visual-based TANs systems recently [10, 20, 21].

Moreover, mobile 2FA is increasingly used by the global service providers such as Google, Twitter and Facebook at user login to mitigate the massive abuse of their services. Users need their login credentials *and* an OTP to complete the login process. The OTPs are sent to the smartphone via SMS messages or over the Internet connection. In addition, some providers offer apps that can generate OTPs on client-side, a convenient setup without the need for out-of-band communication.

Goal, Contributions and Outline. The main goal of our paper is to investigate and evaluate the security of various mobile 2FA schemes that are currently deployed in practice and are used by millions of customers/users.

Single-infection attacks on mobile 2FA schemes. We investigate the deployed mobile 2FA of Google, Twitter, Facebook and Dropbox service providers (Section 3). We point out their conceptual and implementation-specific security weaknesses and show how malware can bypass them, even when a single device, a PC, is infected. For

[4] Also by the world's biggest banks such as Bank of America, Deutsche Bank, Santander in UK, ING in the Netherlands, and ICBC in China.

example, some providers allow the user to deactivate 2FA without the need to verify this transaction with 2FA – an easy way for PC malware to circumvent the scheme. Other providers offer master passwords, which as we show, can be stolen and then be used to authenticate without using an OTP. Moreover, we found a weakness in the OTP generation scheme of Google which reduces the entropy of generated OTPs. We further show how to exploit Google Authenticator, a mobile 2FA login protection app used by dozens of service providers.

A more general 2FA attack based on dual infections: Then we turn our attention to more sophisticated attacks of general nature, and show that even if one of the devices (involved in a 2FA) is infected by malware, it can infect the other device with a *cross-platform infection* in realistic adversary settings (Section 4). We demonstrate the feasibility of such attacks by prototyping PC-to-mobile and mobile-to-PC cross-platform attacks. Our concept significantly enhances the well-known banking Trojans ZeuS/ZitMo [32] or SpyEye/SpitMo [7]. In contrast to these attacks that need to lure users by phishing, our technique does not require any user interaction and is completely stealthy. Once both devices are infected, the adversary can bypass various instantiations of mobile 2FA schemes, which we show by prototyping attacks against SMS-based and visual transaction authentication solutions of banks and login verification schemes of various Internet providers.

2FA malware in the wild. Finally, to underline the importance to redesign mobile 2FA systems, we cluster and reverse engineer hundreds of real-world malicious apps that target mobile 2FA schemes (Section 5). Our analysis confirms, for example, that banking Trojans already deploy mobile counterparts which allow to steal 2FA credentials like TANs.

2 Background

Mobile 2FA schemes can be classified according to (i) what is protected with the second authentication token (the OTP), and (ii) how the OTP is generated.

What does 2FA protect? 2FA schemes are widely deployed in two major application areas of online banking and login authentication. Online banking systems use TANs (Transaction Authentication Numbers) as an OTP to authenticate transactions submitted by the user to the bank. TANs are typically cryptographically bound to the transaction data and are valid only for the given transaction. Recently, 2FA login schemes were also deployed by large Internet service providers such as Google, Apple, Dropbox, Facebook, to name but a few. These systems use OTPs during the user authentication process to mitigate attacks on user passwords, such as phishing and keyloggers.

Where are OTPs generated? OTP can be either generated locally on the client side (e.g., on the mobile device of the user), or by the service provider on server-side with an OTP transfer to the user via an out-of-band (OOB) channel. Client-side OTP generation algorithms may, for example, rely on a shared secret and time synchronization between the authentication server and the client, or on a counter-based state that is shared between the client and the server. This approach allows the OTP to be generated offline, as no communication with the server is required.

In contrast, server-side generated OTPs use OOB channels to transmit an OTP from the server to the client. The most popular *direct* OOB is SMS messaging over cellular networks, which offers a high availability for users, as normally any mobile phone is capable of receiving SMS messages. However, SMS-based services incur additional costs, hence, many service providers propose alternative solutions which use the Internet for direct transmission of the OTP with no additional costs. For example, a mobile app could receive an encrypted OTP from the server over the Internet and then decrypt and display the OTP to the user. As a downside, Internet-based OTP transfers require the customer's phone to be online during the authentication process.

An alternative to online apps is an *indirect* OOB channel between a mobile app and a server via the user's PC. This solution uses the PC's Internet connection to deliver an encrypted OTP from the server to the PC, and a side-channel to transfer the OTP from the PC to the mobile phone for further decryption. For example, the server can generate and encrypt an OTP (or a nonce) and transfer it to the PC in form of a visual cryptogram and display it on a web site[5]. To get this value, a mobile device then scans and decrypts the cryptogram. As the transferred value is encrypted on the server side and decrypted on the mobile device, the PC cannot obtain it in plain text. This solution does not require the mobile phone to be online. In practice, this technique is used by visual TAN solutions which increasingly gain popularity in online banking [10, 20, 21].

3 Single-Infection Attacks on Mobile 2FA

In this section, we analyze the security of mobile 2FA systems in face of compromised computers. We consider mobile 2FA schemes as secure if an adversary who compromised only a user's PC (but has no control over a mobile device) cannot authenticate in the name of the user. Such an attacker model is reasonable, as assuming a trustworthy PC would eliminate the need in utilizing a separate device to handle the secondary authentication credential.

3.1 Low-entropy OTPs

In the following, we analyze the strength of OTPs generated by the four service providers under analysis. In general, low-entropy passwords are vulnerable to brute-force attacks. We thus seek to understand if the generated OTPs fulfill basic randomness criteria. For this, we implemented a process to automatically collect OTPs from Twitter, Dropbox and Google. We had to exclude the Facebook service from this particular test, as our test accounts were blocked after collecting only a few OTPs – presumably to keep SMS-related costs manageable.

To automate the collection process of OTPs, we implemented host software that initiates the login verification and submits the login credentials, while a mobile counterpart monitors incoming SMS messages on the mobile device and extracts OTPs

[5] Alternatively, the server can send a secret value to be used in OTP generation on the client side rather than an OTP itself

Service Provider	Number of collected OTPs	Number of unique OTPs	Collection interval, min.	Average OTP value
Dropbox	1564	1561	15	507809
Google	659	654	30	559851
Twitter	775	772	15	505883

Table 1. Collection of one-time-passwords

into a database. The intercepted OTP is then used to complete the authentication process at the PC. We repeat this procedure periodically. We used a collection time interval of 15 min for Dropbox and Twitter, but had to increase it to 30 min for Google to avoid our account from being blocked. In total, we collected 1564 (Dropbox), 659 (Google) and 775 (Twitter) OTPs. All investigated services create 6-digit OTPs represented in decimal format. We provide the collection details in Table 1 and a graphical representation of the collected OTPs in Appendix A.

While the OTPs generated by Dropbox and Twitter passed standard randomness tests, we observed that Google OTPs never start with a '0' digit. Leaving out 1/10th of all possible OTP values reduces the entropy of the generated passwords, as the number of possible passwords is reduced by 10% from 10^6 to $10^6 - 10^5$.

3.2 Lack of OTP Invalidation

We made another important observation concerning *invalidation* of OTPs. We noticed that – if we do not complete the 2FA process – Google repeatedly created the same OTP for consecutive authentication trials. Google only invalidates OTPs (i) after an hour, or (ii) after a user successfully completed 2FA. We tested that the OTPs repeat even if the IP address, browser and OS version of the user who wants to log in changes. An attacker could exploit this weakness to capture an OTP, while at the same time preventing the user from submitting the OTP to the service provider. This way, the captured OTP remains valid. The adversary can then re-use the OTP in a separate login session, as Google will still expect the same OTP – even for a different session. Similar man-in-the-browser attacks are also possible if OTPs are invalidated, but add a higher practical burden to the attacker.

3.3 2FA Deactivation

If 2FA is used for login verification, users can typically opt-in for the 2FA feature. In the following, we investigate how users (or attackers) can opt-out from the 2FA feature. Ideally, disabling 2FA would require further security checks. Otherwise we risk that PC malware can hijack existing sessions in order to disable 2FA.

We therefore analyzed the deactivation process for the four service providers. We created an account per provider, logged in to these accounts, enabled 2FA and – to delete any session information – signed out and logged in again. We observed that when logged in, users of Google and Facebook services can disable 2FA without any additional authentication. Twitter and Dropbox additionally require user name and password. None of the investigated service providers requested an OTP to authorize this action. Our observations imply that the 2FA schemes of the evaluated providers can be bypassed by PC malware without the need to compromise the mobile device. PC malware can wait until a user logs in, then hijack the session and disable 2FA in the user's account settings. If additional login credentials are required to confirm this operation (as required by Twitter and Dropbox), the PC malware can re-use credentials that can be stolen, e.g., by key logging or by a man-in-the-browser attack.

3.4 2FA Recovery Mechanisms

While 2FA schemes promise improved security, they require users to have their mobile devices with them to authenticate. This issue may affect usability, as users may lose control over their accounts if control over their mobile device is lost (e.g., if the device is lost, stolen or temporarily unavailable due to discharged battery). To address this issue, service providers enable a recovery mechanism which allows users to retain control over their account in absence of their mobile device. On the downside, attackers may misuse the recovery mechanism and be also able to gain control over user's account without compromising user's mobile device.

Among the evaluated providers, Twitter does not provide any recovery mechanism. Dropbox uses a so-called recovery password, a 16-symbols-wide random string in a human-readable format, which appears in the account settings and is created when the user enables 2FA. Facebook and Google use another recovery mechanism. They offer the user an option to generate a list of ten recovery OTPs, which can be used when she has no access to her mobile device. The list is stored in the account settings, similar to the recovery passwords of Dropbox. Dropbox and Google do not require any additional authentication before allowing access to this information, while Facebook additionally asks for the login credentials.

As the account settings are available to users after they logged in, these recovery credentials (OTPs and passwords) can be accessed by malware that hijacks user sessions. For example, a PC-residing malware can access this data by waiting until the user signs in to her account. Hijacking the session, the malware can then obtain the recovery passwords from the web page in the account settings – bypassing the additional check for login credentials (as in the case of Facebook).

3.5 OTP Generator Initialization Weaknesses

Schemes with client-side generated 2FA OTPs, such as Google Authenticator (GA), rely on pre-shared secrets. The distribution process of pre-shared secrets is a valuable attack vector. We analyzed the initialization process of the GA app, which is used by dozens of services including Google Mail, Facebook or Outlook.com.

The GA initialization begins when the user enables GA-based authentication in her account settings. The service provider generates a QR code which is displayed to the user (on the PC) and should be scanned by the user's smartphone. The QR code contains all information necessary to initialize GA with user-specific account details and pre-shared secrets. We analyzed the QR code sent by Facebook and Google during initialization process and identified the structure of the QR code. It includes details as the type of the scheme (counter-based vs. time-based), service and account identifier, a counter (only for counter-based mode), the length of the generated OTP and the shared secret. Further, all this data is presented *in clear text.* To check if any alternative initialization scheme is supported by GA, we reverse engineered the app with the JEB Decompiler and analyzed the app internals. We didn't identify any alternative initialization routines, which indicates that all 32 service providers using GA use this initialization procedure.

Unfortunately, a PC-residing malware can intercept the initialization message (clear text encoded as an QR code). The attacker can then initialize her own version of the GA and can generate valid OTPs for the target account.

4 Dual-Infection Attacks on Mobile 2FA

In this section we use cross-platform infection attacks in the context of mobile 2FA schemes. We show that given one compromised device, either PC or a mobile phone, an attacker is able to compromise another one by launching a cross-platform infection attack. Our proof-of-concept prototypes (Section 4.1) show that such attacks are feasible and, hence, it is not reasonable to exclude them from the adversary model of mobile 2FA schemes. When both 2FA devices are compromised, the attacker can steal both authentication tokens and impersonate the legitimate user, with no matter what particular instantiation of mobile 2FA is used. To support our statement, we implement attacks against different instantiations of mobile 2FA schemes deployed by banks and popular Internet service providers (Section 4.2).

4.1 Cross-platform Infection Attacks

In the following we demonstrate the feasibility of cross-platform attacks by developing two prototypes: PC-to-mobile cross-platform attack in LAN/WLAN networks and mobile-to-PC attack during tethering. We first specify assumptions and then describe corresponding attack scenarios. Our attack implementations target Android 2.2.1 for the mobile device and Windows 7 for the PC. A detailed description of the attack implementation is available in the extended version of this paper [22].

Assumptions We assume that one of the 2FA devices, either PC or a mobile phone, is compromised. This assumption is reasonable given high rate of infected PCs and recent increase in infection rate for mobile devices [39]. Further, it is a state-of-the art assumption for mobile 2FA schemes. We further assume that the second device, either mobile device or PC, suffers from a vulnerability which allows the attacker to gain control over the code execution. The probability for such vulnerabilities is quite high for both, mobile and desktop operating systems. As a reference, the National Vulnerability Database [2] lists more than 55,000 discovered information security vulnerabilities and exposures for mainstream platforms. Despite decades of history, these vulnerabilities are a prevalent attack vector and pose a significant threat to modern systems [40].

PC-to-Mobile Infection in LAN/WLAN Networks LAN/WLAN networks are often used at home, at work or in public places, such as hotels, cafés or airports. Users often connect both, their PCs and mobile devices to the same network (e.g., in home networks). To perform cross-platform infection in the LAN/WLAN network, the malicious device (either the PC or the mobile device, depending on which device was primarily infected) becomes a man-in-the-middle (MITM) between the target device and the Internet gateway in order to infect the target via malicious payloads. To become a MITM, techniques such as ARP cache poisoning [6] or a rogue DHCP server [26] can be used. Next, the MITM supplies an exploit to the victim which results in code injection and remote code execution. For our implementation, we used a rogue DHCP server attack to become a MITM. The PC advertises itself as a network gateway and becomes a MITM when its malicious DHCP configuration is accepted by the mobile device.

As the MITM, the PC can manipulate Internet traffic supplied to the mobile device. When the user opens the browser in his mobile device and navigates to any web page, the request is forwarded to the PC due to the network configuration of the mobile device specifying the PC as a gateway. The malicious PC does not provide the requested page, but supplies a malicious page containing an *exploit* triggering a vulnerability in the web-browser. In our prototype we used a use-after-free vulnerability CVE-2010-1759 in WebKit, the web engine of the Android browser. Further, we perform a privilege escalation to root by triggering the vulnerability CVE-2011-1823 in the privileged Android's volume manager daemon process.

Mobile-to-PC Infection during Tethering Sessions Tethering allows sharing the Internet connection of the mobile device with other devices such as laptops. During tethering sessions the mobile device is mediating the Internet traffic of the PC, hence it is already a MITM and can reply any HTTP request originating from the PC with a malicious web page containing an exploit. In our implementation, we exploited the vulnerability CVE-2012-4681 in JRE that was introduced in Java 7, which allowed us to disable the security manager of Java and achieve execution of arbitrary Java code. Further, to gain privileges sufficient for intercepting login credentials, we additionally exploited a flaw (CVE-2010-3338) in the Windows Task Scheduler that allowed us to elevate privileges to administrative rights.

4.2 Bypassing Different Instantiations of Mobile 2FA Schemes

In the following, we present instantiations of dual-infection attacks against wide range of mobile 2FA schemes. Particularly, we prototyped attacks against SMS-based TAN schemes of several banks, bypassed 2FA login verification systems of popular Internet service providers, defeated the visual TAN authentication scheme of Cronto and circumvented Google Authenticator. Overall, our prototypes demonstrate successful attacks against mobile 2FA solutions of different classes (cf.Section 2).

Schemes with Server-side Generated OTPs and Direct OOB A direct OOB channel between the remote server and the mobile device can be realized either based on HTTPS, or via SMS messages. SMS-based channel is predominating and is widely used for TAN schemes in online banking (e.g., it is deployed by banks in Germany, Spain, Switzerland, Austria, Poland, Holland, Hungary, USA and China). Further, SMS-based OTP-based login verification systems became recently popular and got deployed by a variety of online service providers such as Dropbox, Facebook, Microsoft, Google and Apple.

To bypass these schemes, our malware steals login credentials (i.e., PIN or password) from the computer before they are transferred to the web server of the bank or the service provider. The malware then also obtains the secondary credential, an OTP or TAN, by intercepting SMS messages on the mobile device.

In our attack implementation, we leverage a man-in-the-browser attack to steal the login credential from the PC. Particularly, we use DLL injection[6]to inject a library into the address space of the browser and hook functions to redirect legitimate function calls to the malicious function residing within the injected DLL. In this way, we can intercept function calls containing the user credentials as plaintext parameters, i.e., before they are sent via encrypted HTTPS communication.

In order to intercept SMS messages, our mobile malware acts as a MITM between the GSM modem and the telephony stack of Android and intercepts all SMS messages of interest (so that the user does not receive them), while it forwards all other SMS messages for “normal” use. Furthermore, we implement an SMS-based command & control protocol between the adversary and the mobile device. The protocol can be used to (de)activate interception of OTPs or TANs or to specify the destination of their forwarding.

We successfully evaluated our prototype on online banking deployments of four large international banks that use the SMS-based TAN schemes[7]. We also implemented and successfully tested the attack against the 2FA login verification systems of Dropbox, Facebook, Google and Twitter. Adding further services is little effort for an attacker, showing the conceptual weakness of current server-side generated OTPs.

Schemes with Server-side Generated OTPs and Indirect OOB A prominent example of the scheme with indirect OOB channel are visual TAN solutions, which have been adapted by some large European banks recently [10, 21]. To bypass the scheme, the malware should leak a login credential from the PC. Further, it could either monitor the mobile device for the received cryptogram and steal OTP as soon as it is generated by the app, or steal keys stored within the app. We opted for the latter option, as it does not require the mobile malware to be persistent once the key material is stolen.

We successfully crafted such an attack against the demo version[8] of Cronto visual transaction signing solution – the CrontoSign app (v. 5.0.3). The app stores its keys in a file in the application directory, which can be accessed by our privileged malware. We used the stolen file to replace analogous file on another (assumed to be adversarial) phone with CrontoSign demo app installed. We then used the man-

[6] `http://securityxploded.com/dll-injection-and-hooking.php`

[7] We keep the names of these banks confidential due to responsible disclosure

[8] We stress that we used a publicly available demo version of CrontoSign for our analysis, while commercial versions were not subject of our investigation

in-the-browser attack (as described above) to steal login credentials from the PC and initiated our own login session. We started the transaction, received the cryptogram via the HTTPS connection and scanned it with our adversarial phone. The app produced correct OTP, which was used then to successfully complete authentication.

Schemes with Client-side Generated OTPs Schemes with client-side generated OTPs do not require an OOB channel to transfer the OTP from the server to the client. Instead, an OTP generator produces the same OTP on both the client and the server side.

The generation algorithm is seeded with a secret that is shared between the server and the mobile client. Typically, the shared secret is exchanged via postal mail or is transferred over HTTPS to the user's PC (as used by the GA app; cf. Section 3.5). The generation algorithm further requires a pseudo-random input like a nonce to randomize the output value of each run. The OTP generation algorithms use different nonce values: Some rely on time synchronization between the server and the client and use the time epoch, others use a counter with a shared state, while a third variant utilizes the previously generated OTP as a nonce.

We select Google Authenticator (GA) as our attack target due to its wide deployment. As of Oct 2013, it is used by 32 service providers, among them Google, Microsoft, Facebook, Amazon and Dropbox. The GA app supports counter-based and time-based credential generation algorithms. In either case, it stores all the security-sensitive parameters (such as the seed and a nonce) for the OTP generation in an application-specific database.

To bypass the scheme, our PC-based malware steals login authentication credentials. Our mobile malware also steals the database file stored in the application directory. We copied the database on another mobile device with an installed GA app and were able to generate the same OTPs as the victim.

5 Real-World 2FA Attacks

Until now, we have drafted attacks that enable attackers to circumvent mobile 2FA systems completely automated. In this section, we analyze real-world malware in order to shed light onto how attackers already bypass 2FA schemes in the wild.

5.1 Dataset

Our real-world malware analysis is based on a diverse set of Android malware samples obtained from different sources. We analyze malware from the Malgenome [42] and Contagiodump[9] projects. In addition, we obtain a random set of malicious Android files from VirusTotal. Note that we aim to analyze malware that attacks 2FA schemes. We thus filter on malware that steals SMS messages, i.e., malware that has the permission to read incoming SMS messages. In addition, we only analyze apps which were labeled as malicious by at least five anti-virus vendors. Our resulting dataset consists of 207 unique malware samples.

[9] see `http://contagiominidump.blogspot.de/`

5.2 Malware Analysis Process

We use a multi-step analysis of Android malware samples. First, we dynamically analyze the malware in an emulated Android environment. Dynamic analysis helps us to focus on the malware's behavior when an SMS message is received. Second, to speed up manual static analysis, we cluster the analysis reports to group similar instances. Third, we manually reverse engineer malware samples from each cluster to identify malicious behavior.

Dynamic Malware Analysis We dynamically analyze the malware samples by running each APK file in an emulated Android environment. In particular, we modified the Dalvik Virtual Machine of an Android 2.3.4 system to log method calls (including parameters and return values) within an executed process. We aim to observe malicious behavior when SMS messages are received, i.e., we are not interested in the overall behavior of an app. We therefore trigger this behavior by simulating incoming SMS messages while the malware is executed. To filter on the relevant behavior, the analysis reports contain only the method calls that followed the SMS injection. This way, we highlight code that is responsible for sniffing and stealing SMS messages, while we ignore irrelevant code parts (such as 3rd-party libraries). Also in the case the malware bundles benign code (e.g., a repacked benign app), our analysis report does not contain potentially benign code parts. We stop the dynamic analysis 60 seconds after we injected the SMS message.

The analysis reports consist of tuples with the format:
$rline = \ < cls, method, (p1, ..., p_x), rval >$, whereas cls represents the class name, $method$ is the method name, $rval$ is the return type/value tuple, and p_i is a list of parameter type/value tuples. $rline$ is one line in the report.

Report Clustering We then use hierarchical clustering to group similar reports in order to speed up the manual reverse engineering process. Intuitively, we want to group samples into a cluster if they have a similar behavior when intercepting an SMS message. We define the similarity between to samples as the normalized Jaccard Similarity between two reports A and B, i.e., $sim(A, B) = \frac{|A \cap B|}{|A \cup B|}$, whereas the reports A and B are interpreted as sets of (unordered) report lines. Two report lines are considered equal if the class name, method name, number and type of parameters and return types are equal. We calculate the distances between all malware samples and group them to one cluster if the distance $d = 1 - sim(A, B)$ is lower than a cut-off-threshold of 40%. In other words, two samples are cluster together if they share at least 40% of the method calls when receiving an SMS message.

Classification Given the lack of ground truth for malware labels, we chose to manually assign labels to the resulting clusters. We use off-the-shelf Java bytecode decompilers such as JD-GUI or Androguard to manually reverse engineer each three samples of the 10 largest clusters to classify the malware into families.

5.3 Analysis Results

Clustering of the 207 samples finished in 3 seconds and revealed 21 malware clusters and 45 singletons. We will describe the most prominent malware clusters in the following. Table 2 (see appendix) details the full clustering and classification results.

AndroRAT, a (former open-source) Remote Administration Tool for Android devices, forms the largest cluster in our analysis with 16 unique malware samples. Attackers use the flexibility of AndroRAT to create custom SMS-stealing apps, for example, in order to adapt the C&C network protocol or data leakage channels. Next to AndroRAT, also the app counterparts of the banking Trojans (ZitMo for ZeuS, SpitMo for SpyEye, CitMo for Citadel) are present in our dataset. Except SpitMo.A, these samples leak the contents of SMS messages via HTTP to the botmaster of the banking Trojans. Two SpitMo variants have a C&C channel that allowed to configure the C&C server address or dropzone phone number, respectively.

We further identified four malicious families that forward SMS messages to a hard-coded phone number. We labeled a cluster *RusSteal*, as the malware samples specifically intercept TAN messages with Russian contents. Except RusSteal, none of the families includes code that is related to specific banking Trojans. Instead, the apps blindly steal all SMS messages, usually without further filtering, and hide the messages from the smartphone user. The apps could thus be coupled interchangeably to any PC-based banking Trojan.

Our analysis shows that malware has already started to target mobile 2FA, especially in the case of SMS-based TAN schemes for banks. We highlight that we face a global problem, and next to the Russian-specific Trojans that we found, incidents in many other countries worldwide have been reported [17, 18, 27]. The emergence of toolkits such as AndroRAT will ease the development of malware targeting specific 2FA schemes. Until now, these families largely rely on manual user installation, but as we have shown, automated cross-platform infections are possible. This motivates further research to foster more secure mobile 2FA schemes.

6 Countermeasures and Trade-offs

This section describes countermeasures against the aforementioned attacks.

Dedicated Hardware Tokens Our attacks affect mobile 2FA schemes, while 2FA schemes that rely on dedicated hardware tokens remain intact. Dedicated tokens have a lower complexity than mobile phones and thus provide a smaller attack surface for software-based attacks – although they may still be vulnerable to attacks such as brute-force against the seed value [11] or information leaks from security servers [8]. In addition, hardware tokens have higher deployment costs and scalability issues, especially if users have accounts at several banks they would need multiple tokens.

Secure Out-of-Band Channel An alternative to dedicated hardware tokens is a system utilizing a more secure OOB channel. For example, service providers could use fixed telephony networks as OOB channel for communicating OTPs to customers. Phone devices used in fixed networks do not typically run third party (untrusted) code and do not have feature-rich communication interfaces, hence they are unlikely to be compromised. However, such a solution would limit the mobility of users, and further, devices used in fixed phone networks may undergo technological changes that decrease their security.

Leveraging Secure Hardware on Mobile Platforms A more flexible alternative to dedicated hardware tokens is utilizing general-purpose secure hardware available on mobile devices for OTP protection. For instance, ARM processors feature the ARM TrustZone technology [14] and Texas Instruments processors have the M-Shield security extensions [15]. Furthermore, SIM cards or mobile platforms may include embedded Secure Elements (SE) (e.g., available on NFC-enabled devices) or support removable SEs (e.g., secure memory cards [25] attached via a microSD slot). Such secure hardware can establish a trusted execution environment (TEE) on the device to run security-sensitive code (such as user authentication) in isolation from the rest of the system, such as early approaches in Google Wallet [1] and PayPass [9]. However, most available TEEs are not open to third-party developers. For instance, SEs available on SIM cards are controlled by network operators and processor-based TEEs such as TrustZone and M-Shield are controlled by phone manufacturers. In addition, solutions utilizing SIM-based SEs would be limited to customers of a particular network operator, while secure memory cards can be used only with devices featuring a microSD slot.

Communication Integrity Cross-platform infection attacks as discussed in Section 4.1 can be defeated by deploying standard countermeasures against MITM attacks. For example, one could enforce HTTPS or tunnel HTTP over a remote trusted VPN. However, the former solution would require changes on all Internet servers currently providing HTTP connections (which is infeasible), while the latter solution adds a significant overhead. Moreover, it is not clear which party is trustworthy to host such a proxy. An orthogonal approach is to have logically disjoint networks (e.g., via VLANs) for mobile devices and stationary computers, so that the mobile devices cannot communicate with the user PCs and vice versa and some infection scenarios (such as our DHCP-based attack) fail accordingly.

Detection of Suspicious Mobile Apps SMS-stealing apps exhibit suspicious characteristics or behavior that can be detected by defenders. For example, with static analysis it may be possible to classify suspicious sets of permissions or to identify receivers for events of incoming SMS messages [43], but only if the malicious code is not dynamically loaded. Similarly, taint tracking may help to detect information leakage [23], but adds a significant overhead and can be evaded with implicit data flows [28]. Using behavioral analysis, one could detect SMS receivers that consume or forward TAN-related SMS. More strictly, one could even disable the feature of consuming SMS messages in the mobile OS so that an attacker cannot hide the

SMS messages that he triggered. However, all these security measures need to consider that the attacks we described are not limited to run in user space. For example, we have shown that we can steal OTPs before any app running in user space is noticed about events such as an incoming SMS message. Consequently, the aforementioned solutions can in principle be evaded by attackers, similarly to the arms-race as in PC-based anti-virus systems.

7 Related Work

In this section we survey previous research on mobile 2FA schemes, on attacks against SMS-based TAN systems and on cross-platform infections.

Mobile 2FA schemes Balfanz et al. [16] aim to prevent misuse of the smartcard plugged into the computer by malware without user knowledge. They propose replacing the smartcard with a trusted handheld device which asks the user for permission before performing sensitive operations. Aloul et al. [12, 13] utilize a trusted mobile device as an OTP generator or as a means to establish OOB communication channel to the bank (via SMS). Mannan et al. [29] propose an authentication scheme that is tolerant against session hijacking, keylogging and phishing. Their scheme relies on a trusted mobile device to perform security-sensitive computations. Starnberger et al. [37] propose an authentication technique called QR-TAN which belongs to the class of visual TAN solutions. It requires the user to confirm transactions with the trusted mobile using visual QR barcodes. Clarke et al. [19] propose to use a trusted mobile device with a camera and OCR as a communication channel to the mobile. The Phoolproof phishing prevention solution [33] utilizes a trusted user cellphone in order to generate an additional token for online banking authentication.

All these solutions assume that the user's personal mobile device is trustworthy. However, as we showed in this paper, an attacker controlling the user's PC can also infiltrate her mobile device by mounting a cross-platform infection attack, which undermines the assumption on trustworthiness of the mobile phone.

Attacks on SMS-based TAN authentication Mulliner et al. [30] analyze attacks on OTPs sent via SMS and describe how smartphone Trojans can intercept SMS-based TANs. They also describe countermeasures against their attack, such a dedicated OTP channels which cannot be easily intercepted by normal apps. Their attack and countermeasure rely on the assumption that an attacker has no root privileges, which we argue is not sufficiently secure in the adversary setting nowadays. Schartner et al. [35] present an attack against SMS-based TAN solutions for the case when a single device, the user's mobile phone, is used for online banking. The presented attack scenario is relatively straightforward as the assumption of using a single device eliminates challenges such as cross-platform infection or a mapping of devices to a single user. Many banks already acknowledge this vulnerability and disable TAN-based authentication for customers who use banking apps.

Cross-platform infection The first malware spreading from smartphone to PC was discovered in 2005 and targeted Symbian OS [3]. Infection occurred as soon as the phone's memory card was plugged into the computer. Another example of cross-platform infection from PC to the mobile phone is a proof-of-concept malware which had been anonymously sent to the Mobile Antivirus Research Association in 2006 [24, 34]. The virus affected the Windows desktop and Windows Mobile operating systems and spread as soon as it detected a connection using Microsoft's ActiveSync synchronization software. Another well-known cross-platform infection attack is a sophisticated worm Stuxnet [31] which spreads via USB keys and targets industrial software and equipment. Further, Wang et al. [41] investigated phone-to-computer and computer-to-phone attacks over USB targeting Android. They report that a sophisticated adversary is able to exploit the unprotected physical USB connection between devices in both directions. However, their attack relies on additional assumptions, such as modifications in the kernel to enable non-default USB drivers on the device, and non-default options to be set by the user.

Up to now, most cross-system attacks were observed in public networks, such as malicious WiFi access points [5] or ad-hoc peers advertising free public WiFi [4]. When a victim connects to such a network, it gets infected and may start advertising itself as a free public WiFi to spread. In contrast to our scenario, this attack mostly affects WiFi networks in public areas and targets devices of other users rather than a second device of the same user. Moreover, it requires user interaction to join the discovered WiFi network. Finally, the infection does not spread across platforms (i.e., from PC to mobile or vice versa), but rather affects similar systems.

8 Conclusion

In this paper, we studied the security of mobile two-factor authentication (2FA) schemes that have received much attention recently and are deployed in security sensitive applications such as online banking and login verification.We identified various ways to evade 2FA schemes without obtaining access to the secondary authentication token (such as one-time passwords) handled on the mobile device. The providers can fix these weaknesses by redesigning the 2FA integration into their services. However, we go beyond that and show a more generic and fundamental attack against mobile 2FA schemes by using cross-platform infection for subverting control over *both* end points involved in the authentication protocol (such as PC and a mobile device). We demonstrate practical attacks on SMS-based TAN schemes of four banks, the visual TAN scheme, SMS-based login verification schemes of Google, Dropbox, Twitter and Facebook, and the 2FA scheme based on the popular Google Authenticator app – showing the generality of the problem.

Our results show that current mobile 2FA have conceptual weaknesses, as adversaries can intercept the OTP transmission or steal private key material for OTP generation. We thus see a need for further research on more secure mobile 2FA schemes that can withstand today's sophisticated adversary models in practice.

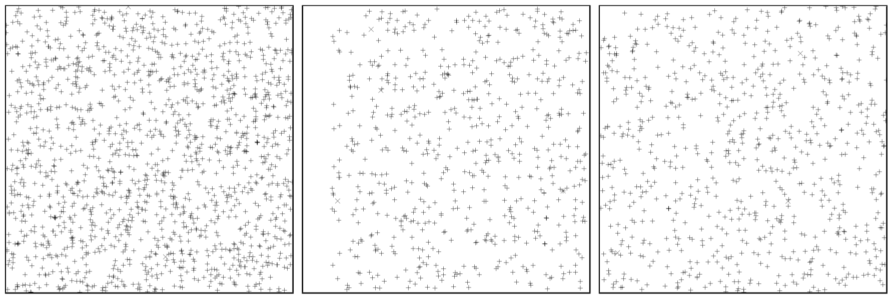
References

1. Google Wallet. http://www.google.com/wallet/how-it-works/index.html.
2. National vulnerability database version 2.2. http://nvd.nist.gov/.
3. Cell phone virus tries leaping to PCs. http://news.cnet.com/Cell-phone-virus-tries-leaping-to-PCs/2100-7349_3-5876664.html?tag=mncol;txt, 2005.
4. The security risks of Free Public WiFi. http://searchsecurity.techtarget.com.au/news/2240020802/The-security-risks-of-Free-Public-WiFi, 2009.
5. KARMA demo on the CBS early show. http://blog.trailofbits.com/2010/07/21/karma-demo-on-the-cbs-early-show/, 2010.
6. Anatomy of an ARP poisoning attack. http://www.watchguard.com/infocenter/editorial/135324.asp, 2011.
7. New Spitmo banking Trojan attacks Android users. http://www.securitynewsdaily.com/1048-spitmo-banking-trojan-attacks-android-users.html, 2011.
8. RSA breach leaks data for hacking securid tokens. http://www.theregister.co.uk/2011/03/18/rsa_breach_leaks_securid_data/, 2011.
9. MasterCard PAYPASS. http://www.mastercard.us/paypass.html#/home/, 2012.
10. Raiffeisen PhotoTAN. http://www.raiffeisen.ch/web/phototan, 2012.
11. RSA SecurID software token cloning: a new how-to. http://arstechnica.com/security/2012/05/rsa-securid-software-token-cloning-attack/, 2012.
12. F. Aloul, S. Zahidi, and W. El-Hajj. Two factor authentication using mobile phones. In *IEEE/ACS Computer Systems and Applications*, May 2009.
13. F. Aloul, S. Zahidi, and W. ElHajj. Multi factor authentication using mobile phones. *International Journal of Mathematics and Computer Science*, 4, 2009.
14. T. Alves and D. Felton. TrustZone: Integrated hardware and software security. *Information Quaterly*, 3(4), 2004.
15. J. Azema and G. Fayad. M-Shield mobile security technology: Making wireless secure. http://focus.ti.com/pdfs/wtbu/ti_mshield_whitepaper.pdf.
16. D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *USENIX Security Symposium - Volume 8*. USENIX Association, 1999.
17. Carlos Castillo, McAfee. Android banking Trojans target Italy and Thailand. http://blogs.mcafee.com/mcafee-labs/android-banking-trojans-target-italy-and-thailand/, 2013.
18. Carlos Castillo, McAfee. Phishing attack replaces Android banking apps with malware. http://blogs.mcafee.com/mcafee-labs/phishing-attack-replaces-android-banking-apps-with-malware, 2013.
19. D. E. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. v. Dijk, S. Devadas, and R. L. Rivest. The untrusted computer problem and camera-based authentication. In *International Conference on Pervasive Computing*. Springer-Verlag, 2002.
20. Cronto Limited. Commerzbank and Cronto launch secure online banking with photoTAN – World's first deployment of visual transaction signing mobile solution. http://www.cronto.com/download/Cronto_Commerzbank_photoTAN.pdf, 2008.
21. Cronto Limited. CorpBanca and Cronto secure online banking transactions with CrontoSign. http://www.cronto.com/corpbanca-cronto-secure-online-banking-transactions-crontosign.htm, 2011.
22. A. Dmitrienko, C. Liebchen, C. Rossow, and A.-R. Sadeghi. On the (in)security of mobile two-factor authentication. Technical Report TUD-CS-2014-0029. CASED. http://www.trust.informatik.tu-darmstadt.de/fileadmin/user_upload/Group_TRUST/PubsPDF/TUD-CS-2014-0029.pdf, 2014.

23. 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 *USENIX OSDI*, 2010.
24. J. Evers. Virus makes leap from PC to PDA. `http://news.cnet.com/2100-1029_3-6044457.html`, 2006.
25. Giesecke & Devrient . The Mobile Security Card offers increased security. `http://www.gd-sfs.com/the-mobile-security-card/mobile-security-card-se-1-0/`.
26. Y. I. Jerschow, C. Lochert, B. Scheuermann, and M. Mauve. CLL: A cryptographic link layer for local area networks. In *International conference on Security and Cryptography for Networks*. Springer-Verlag, 2008.
27. E. Kalige and D. Burkey. Eurograbber: How 36 million euros was stolen via malware. `http://www.cs.stevens.edu/~spock/Eurograbber_White_Paper.pdf`.
28. D. King, B. Hicks, M. Hicks, and T. Jaeger. Implicit flows: Can't live with Em, can't live without Em. In *Information Systems Security*. Springer, 2008.
29. M. Mannan and P. C. Van Oorschot. Using a personal device to strengthen password authentication from an untrusted computer. In *FC'07/USEC'07*, 2007.
30. C. Mulliner, R. Borgaonkar, P. Stewin, and J.-P. Seifert. SMS-based one-time passwords: Attacks and defense (short paper). In *DIMVA*, July 2013.
31. N. Falliere. Exploring Stuxnet's PLC infection process. `http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process`, 2010.
32. V. News. Teamwork: How the ZitMo Trojan bypasses online banking security. `http://www.kaspersky.com/about/news/virus/2011/Teamwork_How_the_ZitMo_Trojan_Bypasses_Online_Banking_Security`, 2011.
33. B. Parno, C. Kuo, and A. Perrig. Phoolproof phishing prevention. In *In Financial Cryptography and Data Security*. Springer-Verlag, 2006.
34. C. Peikari. Analyzing the crossover virus: The first PC to Windows handheld cross-infector. `http://www.informit.com/articles/article.aspx?p=458169`, 2006.
35. P. Schartner and S. Bürger. Attacking mTAN-applications like e-banking and mobile signatures. Technical report, University of Klagenfurt, 2011.
36. Sparkasse. Online banking mit chipTAN. `https://www.sparkasse-pm.de/privatkunden/banking/chiptan/vorteile/index.php?n=/privatkunden/banking/chiptan/vorteile/`.
37. G. Starnberger, L. Froihofer, and K. Goeschka. QR-TAN: Secure mobile transaction authentication. In *ARES*. IEEE, 2009.
38. A. S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, 2001.
39. TrendLabs. 3Q 2012 security roundup. Android under siege: Popularity comes at a price. `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/reports/rpt-3q-2012-security-roundup-android-under-siege-popularity-comes-at-a-price.pdf`, 2012.
40. V. van der Veen, N. dutt Sharma, L. Cavallaro, and H. Bos. Memory errors: The past, the present, and the future. In *RAID*, 2012.
41. Z. Wang and A. Stavrou. Exploiting smart-phone USB connectivity for fun and profit. In *26th Annual Computer Security Applications Conference*. ACM, 2010.
42. Y. Zhou and X. Jiang. Dissecting Android malware: Characterization and evolution. In *IEEE Symposium on Security and Privacy*, 2012.
43. 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 *NDSS*, 2012.

Appendix A Graphical Representation of OTPs

We plot a 6-digit OTP by plotting its two halves on the x- and y-axis (1000 dots wide). For example, the OTP "012763" is plotted at x=12 and y=763. Symbols '+' and 'x' represent one and two occurrences of the same OTP, respectively. Empty space at the left side of Figure 1(b) means that Google OTPs never start with a '0' digit.



(a) Dropbox (b) Google (c) Twitter

Fig. 1. Collected OTPs from three service providers

Appendix B Mobile Malware Clustering Results

Family	**C&C**	**Leaked TAN via**	**# Samples**
AndroRAT	TCP	TCP	16
ZitMo.A	n/a	HTTP (GET)	13
SpitMo.A	SMS	SMS	13
Obfake.A	n/a	SMS	12
SpitMo.C	HTTP	HTTP (GET)	6
RusSteal	n/a	SMS	6
Koomer	n/a	SMS	5
Obfake.B	n/a	SMS	4
SpitMo.B	n/a	HTTP (POST)	3
CitMo.A	n/a	HTTP (GET)	3

Table 2. Real-world malware families targeting 2FA by stealing SMS messages