SECURITY ANALYSIS OF MOBILE TWO-FACTOR AUTHENTICATION SCHEMES

Contributors

Alexandra Dmitrienko
Fraunhofer SIT

Christopher Liebchen
Technische Universität Darmstadt

Christian Rossow
Vrije Universiteit Amsterdam

Ahmad-Reza Sadeghi
Intel Collaborative Research Institute for Secure Computing

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 (such as a smartcard) to store and handle the secondary authentication token, and hence are considered as a reasonable tradeoff between security, usability, and cost. They are widely used in online banking and increasingly deployed by Internet service providers.

In this article, 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 2FA, even when the secondary authentication token is not under the attacker's control. We then go a step further and present a more general attack against mobile 2FA schemes. Our attack relies on a 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.

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 logins to personal accounts. Two-factor authentication (2FA) schemes promise a higher protection level by extending the single authentication factor, that is, *what the user knows*, with other authentication factors such as *what the user has* (for example, a hardware token or a smartphone), or *what the user is* (for example, biometrics).[29]

"…the traditional login/password authentication is considered insufficiently secure for many security-critical applications such as online banking or logins to personal accounts."

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

While 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[27] 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 (for example, for accounts at several banks). On the other hand, 2FA schemes that use mobile devices (such as smartphones) have become popular recently and have been adopted by many banks and large service providers. These *mobile 2FA* schemes are considered to provide an appropriate tradeoff between security, usability, and cost, and are the focus of this article.

"These mobile 2FA schemes are considered to provide an appropriate tradeoff between security, usability, and cost…"

A prominent example of mobile 2FAs are SMS-based TAN systems (known as mTAN, smsTAN, or mobileTAN). Their goal is to mitigate account abuse even if the banking login credentials have been compromised, for example, by a PC-based banking Trojan. Here, the service provider (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, 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. Further, some large European banks have adopted visually based TAN systems recently.[7][14][15] Moreover, mobile 2FA is increasingly used by the global service providers such as Google, Twitter, and Facebook 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 the client side, a convenient setup without the need for out-of-band communication. For instance, such an approach is followed by Google Authenticator, the popular 2FA app currently used by 32 third-party service providers.

"…mobile 2FA is increasingly used by the global service providers such as Google, Twitter, and Facebook to mitigate the massive abuse of their services."

Goal, Contributions, and Outline

The main goal of our article 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 (see the next section, "Single-Infection Attacks on Mobile 2FA"). 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 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. 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 (see the section "Dual-Infection Attacks on Mobile 2FA"). We demonstrate the feasibility of such attacks by prototyping PC-to-mobile cross-platform attacks. Our concept significantly enhances the well-known banking Trojans ZeuS/ZitMo[23] or SpyEye/SpitMo.[6] 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.

"…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…"

- *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 (see the section "Real-World 2FA Attacks"). Our analysis confirms, for example, that banking Trojans already deploy mobile counterparts that allow attackers to steal 2FA credentials like TANs.

"…banking Trojans already deploy mobile counterparts that allow attackers to steal 2FA credentials like TANs."

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.

Low-Entropy OTPs

Here 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 exhibit full 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, because 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 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 minutes for Dropbox and Twitter, but had to increase it to 30 minutes 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 graphical representation of the collected OTPs in Figure 1.

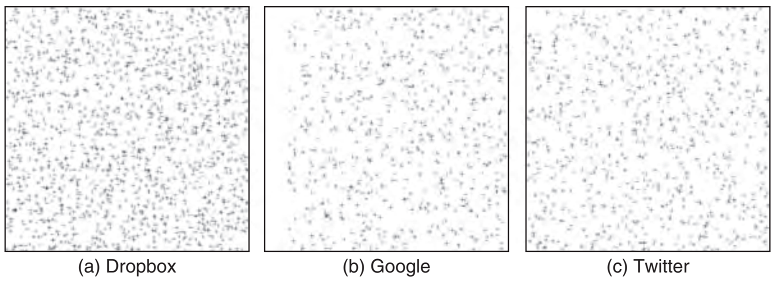

Figure 1: OTPs collected from three service providers. We plot a 6-digit OTP by plotting its two halves on the *x*- and *y*-axis (1000 dots wide). For example, the OTP "012763" is plotted at $x = 12$ and $y = 763$. Symbols "+" and "×" represent one and two occurrences of the same OTP, respectively.
(Source: Dmitrienko, Liebchen, Rossow, and Sadeghi, 2014)

While the OTPs generated by Dropbox and Twitter passed standard randomness tests, we observed that Google OTPs never start with a zero. Leaving out one tenth of all possible OTP values reduces the entropy of the generated passwords: the number of possible passwords is reduced by 10 percent from 10^6 to $10^6 - 10^5$.

"…we observed that Google OTPs never start with a zero."

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.

"We noticed that—if we do not complete the 2FA process—Google repeatedly created the same OTP for consecutive authentication trials."

The adversary can then reuse the OTP in a separate login session, because 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 they add a higher practical burden to the attacker.

2FA Deactivation

If 2FA is used for login verification, users can typically opt in for the 2FA feature. In the following section, 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 might hijack existing sessions in order to disable 2FA.

We therefore analyzed the deactivation process for the four service providers. We created one 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, and 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 reuse credentials that can be stolen, for example, by applying key logging or a man-in-the-browser attack.

"PC malware can wait until a user logs in, and then hijack the session and disable 2FA in the user's account settings."

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, because users may lose control over their accounts if control over their mobile device is lost (for example, if the device is lost, stolen, or temporarily unavailable due to discharged battery). To address this issue, service providers enable recovery mechanisms that allow users to retain control over their account in the absence of their mobile device. On the downside, attackers may misuse the recovery mechanism in order to gain control over user accounts without compromising the mobile device.

"…attackers may misuse the recovery mechanism in order to gain control over user accounts without compromising the mobile device."

Among the evaluated providers, Twitter does not provide any recovery mechanism. Dropbox uses a so-called recovery password, a 16-symbol-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 users an option to generate a list of ten recovery OTPs, which can be used when they have no access to their 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 have logged in, these recovery credentials (OTPs and passwords) can be accessed by malware that hijacks user sessions. For example, PC-residing malware can access this data by waiting until users sign in to their 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).

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, and Outlook.com.

The GA initialization begins when the user enables GA-based authentication in the user's account settings. The service provider generates a QR code that 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 the initialization process and identified the structure of the QR code. It includes such 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. 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 did not identify any alternative initialization routines, which indicates that all 32 service providers using GA use this initialization procedure.

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

"…PC-residing malware can intercept the initialization message (clear text encoded as a QR code). The attacker can then initialize the attacker's own version of the GA and can generate valid OTPs…"

Dual-Infection Attacks on Mobile 2FA

In this section, we present a more general attack against mobile 2FA schemes. Particularly, we present the attack model that does not rely on implementation weaknesses (as, for example, weaknesses reported in the previous section), but rather conceptual. Particularly, we apply cross-platform infection attacks (PC-to-mobile) in context of mobile 2FA schemes. Our attack model undermines the security of a large class of 2FA schemes that are widely used in security-critical applications such as online banking and login verification.

"…we apply cross-platform infection attacks (PC-to-mobile) in context of mobile 2FA schemes."

System Model

Our system model is depicted in Figure 2. It includes the following actors: (i) a user U, (ii) a web server S, (iii) a computer C, (iv) a mobile device M, and (v) a remote malicious server A. The user U is a customer who is subscribed for the online service. The web server S is maintained by the service provider of the online service. The computer C is either a desktop PC or a laptop used by the user to access the web site hosted by S. The mobile device M is a handheld computer or a smartphone of U, which is involved in authentication of U against S.

The legitimate communication between the entities is illustrated with dashed arrows in Figure 2. To get access to the service, U has to prove to S possession of both authentication tokens T1 and T2. The first authentication token T1 is handled by C (typically represented by login credentials). The second authentication token T2 is handled by the mobile device M. T2 is an OTP which is either received from S via an out-of-band channel, or generated locally on M.

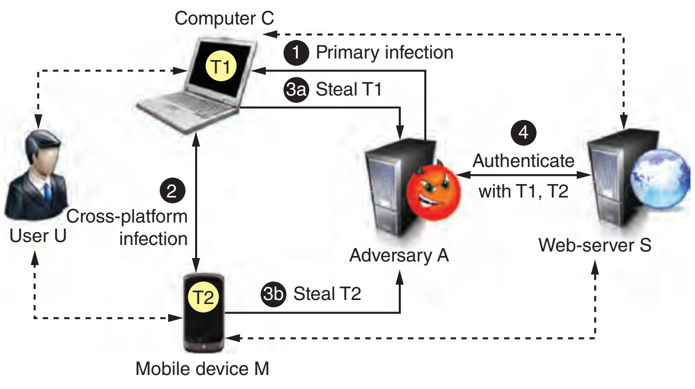

Figure 2: System model and attack steps
(Source: Dmitrienko, Liebchen, Rossow, and Sadeghi, 2014)

A remote malicious server A represents an adversary who aims to gain control over C and M and to steal authentication tokens T1 and T2 in order to be able to successfully authenticate against S in the name of U.

"…nowadays many PCs are infected."

Assumptions

We assume that *C*, the user's PC, is compromised. This assumption is reasonable, because nowadays many PCs are infected. We further assume that the second device, either M or C, suffers from a (memory-related) vulnerability that allows the attacker to subvert the 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[1] 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.[31]

"Despite decades of history, these vulnerabilities are a prevalent attack vector and pose a significant threat to modern systems."

Attack Description

The general attack scenario has four phases, which are illustrated by solid lines in Figure 2: (i) primary infection; (ii) cross-platform infection; (iii) stealing authentication tokens, and (iv) authentication.

1. *Primary infection*. We do not specify the way the attacker achieves a primary infection. Instead, we assume that C is already infected (see the previous section, "Assumptions").

2. *Cross-platform infection.* The infected C attempts to compromise M by triggering a memory-related vulnerability. Exploitation is possible if, for example, both devices are connected to a single network, as described in the following section, "Cross-Platform Infection in LAN/WLAN Networks."
3. *Stealing authentication tokens.* As we will show, when controlling M and C, an attacker A can obtain both authentication tokens T1 and T2 (steps 3a and 3b respectively). Static authentication tokens that do not change from one to another authentication session (such as login credentials) are immediately transmitted to and persistently stored at A.
4. *Authentication.* Authentication is performed by A, who controls both authentication tokens. A has a local copy of static authentication tokens (such as login credentials), and can obtain OTPs by forwarding them from M to A. Note that A does not only hijack the session of U, but can even establish the attacker's own sessions at any time and independently from U.

Cross-Platform Infection in LAN/WLAN Networks

LAN/WLAN networks are often used at home, at work, or in public places, such as hotels, cafes, or airports. Users often connect both their PCs and mobile devices to the same network (for example, in home networks). To perform cross-platform infections in the LAN/WLAN network, the malicious PC becomes a man in the middle (MITM) between the mobile device and the Internet gateway in order to infect it via malicious payloads. To become an MITM, techniques such as ARP cache poisoning[5] or a rogue DHCP server[18] can be used. Next, the MITM supplies an exploit to the victim, which results in code injection and remote code execution.

"Users often connect both their PCs and mobile devices to the same network…"

For our implementation of cross-platform infection, we used a rogue DHCP server attack to become an MITM. In particular, C advertises itself as a network gateway and becomes an MITM when its malicious DHCP configuration is accepted by M. As the MITM, C can manipulate Internet traffic supplied to M. When M connects to the network and requests an IP address, this request is served by our malicious DHCP server, which assigns a valid configuration for this network, but substitutes the correct gateway IP address with its own. The malware loads a driver that implements network address translation (NAT) to dynamically forward any HTTP request to an external or local HTTP server. This server answers every HTTP request with a malicious web page.

When U opens the browser in M and navigates to any web page, the request is forwarded to C due to the network configuration of M specifying C as a gateway. The malicious C does not provide the requested page, but supplies a malicious page containing an exploit triggering the 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. We further perform a privilege escalation to root by triggering the vulnerability CVE-2011-1823 in the privileged Android's volume manager daemon process.

"The malicious C does not provide the requested page, but supplies a malicious page containing an exploit triggering the vulnerability in the web browser."

Bypassing Different Instantiations of Mobile 2FA Schemes

Next we present instantiations of dual-infection attacks against a 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.

"…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."

Bypassing SMS-based TAN Schemes and 2FA Login Verification Schemes

To bypass SMS-based TAN schemes used by banks and 2FA login verification systems, we launched a man-in-the-browser attack on the PC to steal the login credentials (that is, PIN or password) from the computer before they are transferred to the web server of the bank or the service provider. Further, we implemented mobile malware that obtains the secondary credential, an OTP or TAN, by intercepting SMS messages on the mobile device. It acts as a man-in-the-middle 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.

"We successfully evaluated our prototype on online banking deployments of four large international banks…"

We successfully evaluated our prototype on online banking deployments of four large international banks (the names of the banks are kept undisclosed) and evaluated it against the 2FA login verification systems of Dropbox, Facebook, Google, and Twitter.

Bypassing Visual TAN Solutions

To demonstrate the effectiveness of dual-infection attacks against visual TAN solutions, we successfully crafted such an attack against the demo version of the Cronto visual transaction signing solution—the CrontoSign app (v. 5.0.3). We reused the man-in-the-browser attack to leak login credentials from the PC and used our mobile malware to steal key material stored by the CrontoSign app. We then copied stolen files with key material onto another (adversarial) phone with CrontoSign installed and then performed a login attempt with stolen login credentials and the adversarial phone. The app on the adversarial phone produced correct OTP, which was then used to successfully complete authentication.

Bypassing Google Authenticator (GA) App

We selected Google Authenticator (GA) as our attack target due to its wide deployment. As of October 2013, it was being used by 32 service providers, among them Google, Microsoft, Facebook, Amazon, and Dropbox. The GA app does not receive OTP from the server, but instead generates it on client side. The generation algorithm is seeded with a secret that is shared between the server and the mobile client and further requires a pseudo-random input like a nonce to randomize the output value of each run. GA supports the following nonce values: shared time (in a form of the time epoch) or a counter with a shared state. In either case, it stores all security-sensitive parameters (such as the seed and a nonce) for the OTP generation in an application-specific database. Hence, to bypass the scheme, our PC-based malware steals login authentication credentials, while our mobile malware leaks the database file stored in the GA 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.

Real-World 2FA Attacks

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

"…we analyze real-world malware in order to shed light onto how attackers already bypass 2FA schemes in the wild."

Dataset

Our real-world malware analysis is based on a diverse set of Android malware samples obtained from different sources.

We analyzed malware from the Malgenome[33] and Contagiodump[34] projects. In addition, we obtained a random set of malicious Android files from VirusTotal. Note that we aimed to analyze malware that attacks 2FA schemes. We thus filtered on malware that steals SMS messages, that is, malware that has the permission to read incoming SMS messages. In addition, we only analyzed apps that were labeled as malicious by at least five antivirus vendors. Our resulting dataset consists of 207 unique malware samples.

Malware Analysis Process

We used a multistep analysis of Android malware samples, as depicted in Figure 3. First, we dynamically analyzed the malware in an emulated Android environment. Dynamic analysis helped us to focus on the malware's behavior when an SMS message is received. Second, to speed up manual static analysis, we clustered the analysis reports to group similar instances. Third, we manually reverse engineered malware samples from each cluster to identify malicious behavior.

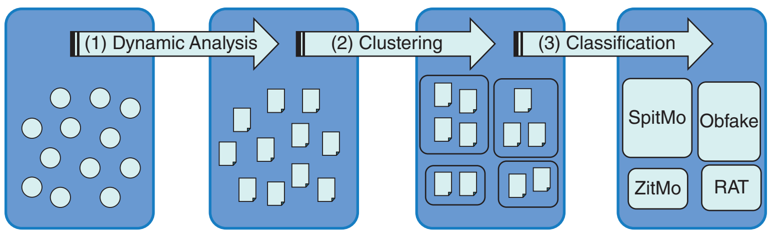

Figure 3: Multistep malware analysis procedure
(Source: Dmitrienko, Liebchen, Rossow, and Sadeghi, 2014)

Dynamic Malware Analysis
We dynamically analyzed 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 aimed to observe malicious behavior when SMS messages were received, that is, we were not interested in the overall behavior of an app. We therefore triggered this behavior by simulating incoming SMS messages while the malware was 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 third-party libraries).

Likewise, in the case the malware bundles benign code (such as a repacked benign app), our analysis report does not contain potentially benign code parts. We stopped the dynamic analysis 60 seconds after we injected the SMS message.

The analysis reports consist of tuples with the format:

rline = <cls, method, (p[1], . . . , 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 used hierarchical clustering to group similar reports in order to speed up the manual reverse engineering process. Intuitively, we wanted to group samples into a cluster if they had a similar behavior when intercepting an SMS message.

We defined the similarity between to samples as the normalized Jaccard similarity between two reports A and B:

$\mathrm{sim}(A, B) = |A \cap B| \,/\, |A \cup B|$,

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 calculated the distances between all malware samples and grouped them to one cluster if the distance $d = 1 - sim\ (A, B)$ is lower than a cutoff threshold of 40 percent. In other words, two samples were clustered together if they shared at least 40 percent 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.

Analysis Results

This section shows the clustering results and gives a detailed analysis of one of the analyzed ZitMo samples.

Clustering Results

Clustering of the 207 samples finished in 3 seconds and revealed 21 malware clusters and 45 singletons.

"Clustering of the 207 samples finished in 3 seconds and revealed 21 malware clusters and 45 singletons."

We now describe the most prominent malware clusters. Table 1 details full clustering and classification results.

Family	Command & Control	Leaked TAN via	# Samples
AndroRAT	TCP	TCP	16
ZitMo.A	SMS	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 1: Real-world malware families targeting 2FA by stealing SMS messages
(Source: Dmitrienko, Liebchen, Rossow, and Sadeghi, 2014)

AndroRAT, a (formerly 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 command and control (C&C) network protocol or data leakage channels.

Next to AndroRAT, the app counterparts of the banking Trojans (ZitMo for ZeuS, SpitMo for SpyEye, CitMo for Citadel) are also 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 the configuration of 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 with any PC-based banking Trojan.

"…the apps blindly steal all SMS messages, usually without further filtering, and hide the messages from the smartphone user."

"…malware has already started to target mobile 2FA, especially in the case of SMS-based TAN schemes for banks."

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.[11][12][19] The emergence of toolkits such as AndroRAT will ease the development of malware targeting specific 2FA schemes.

Until now, these families have largely relied 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.

ZitMo Case Study

We now outline the reverse engineering results of one of the samples to show the inner workings of real-world malware in more detail. Here we provide a case study on the ZitMo malware samples (we analyzed the ZitMo sample with a SHA256 value of ceb54cba2561f62259204c39a31dc204105d358a1a10cee37de889332fe6aa27), which are the mobile counterparts of the ZeuS banking Trojan.

In order to install ZitMo, the ZeuS Trojan manipulates an online banking session such that ZeuS-infected users are asked to enter their mobile phone number. Once they do so, the attackers send an SMS message with a link to *security software*, which in fact is a camouflaged ZitMo Trojan. In contrast to the attack that we have described, the infection of the mobile device is a largely manual process and requires user interaction.

Once ZitMo is installed, it asks the user to enter a verification code, which the attackers use to establish a unique mapping between the infected PC and the mobile counterpart. From this point on, ZitMo operates in background. As ZitMo has registered as a broadcast receiver for SMS messages, it can intercept, manipulate, and read all incoming SMS messages.

Whenever an SMS message is received, ZitMo first checks if it contains a hidden command that can be used to reconfigure ZitMo. Such messages remain hidden to the user: they are not visible in the default messaging app. ZitMo embeds the content of all other messages as HTTP request parameters and sends the data (including the device ID) to the ZitMo dropzone server. Before the data can be forwarded, ZitMo needs to reverse its obfuscation of the dropzone URL. If the HTTP request fails, ZitMo stores the message in hidden data storage and retries submission at a later stage.

"…ZitMo or similar mobile malware have been observed to steal tens of millions of dollars from infected users."

Although using a simple scheme, ZitMo or similar mobile malware have been observed to steal tens of millions of dollars from infected users.[19]

Countermeasures and Tradeoffs

Possible defense strategies against attacks on mobile 2FA schemes can be divided into two classes: preventive and reactive countermeasures. Preventive countermeasures, such as exploitation mitigation, OS-level security extensions, leveraging secure hardware, and using trusted VPN proxies, are applied in order to reduce the attack surface, while reactive countermeasures aim to detect ongoing attacks in order to mitigate further damage.

Exploitation Mitigation

Our cross-device infection attack relies on exploitation of memory-related vulnerability (see the earlier section, "Assumptions"), hence, mitigation techniques against runtime exploitations would be an effective countermeasure. However, despite more than two decades of research, such flaws still undermine security of modern computing platforms.[31] Particularly, while the Write-XOR-Execute (W^X)[37] security model prevents code injection (enforced on Android since 2.3 version), it can be bypassed by code reuse attacks, such as ret2libc[38] or return-oriented programming (ROP).[39] Code reuse attacks do not require code injection, but rather invoke execution of functions or sequences of instructions that already reside in the memory. Because code reuse attacks make use of memory addresses to locate instruction sequences to be executed during the attack, the mitigation techniques were developed that randomize program memory space, making it hard to predict exact addresses prior to program execution. For instance, address space layout randomization (ASLR)[40], which adds a random offset to loaded code at each program start, is available on iOS starting from version 4.3 and was also recently introduced for Android (in 4.0 version).

However, ASLR can be bypassed by brute-forcing the offset at runtime[41], which generated a new line of research on fine-grained address space randomization[42][43][44][45][46][47] (down to instruction level), which makes brute-force attacks infeasible. Unfortunately, fine-grained address space randomization techniques are ineffective in the presence of memory disclosure bugs. Particularly, these bugs can be utilized to disclose memory content and build a return-oriented programming (ROP) payload dynamically at runtime.[48,49]

Hence, while the deployed memory mitigation techniques raise the bar for the type of cross-device infection we demonstrated, such attacks are still possible, even if all protections are enforced.

"…while the deployed memory mitigation techniques raise the bar for the type of cross-device infection we demonstrated, such attacks are still possible…"

OS Level Security Extensions

OS security improves over time and can mitigate some attack classes. With respect to the threat of mobile malware targeting 2FA, the first significant changes appeared in version 4.2 of Android, where a new system API was introduced allowing users to verify and to selectively grant or deny permissions requested by applications during application installation. Ideally, the users can choose during the installation process what privileges a (potentially malicious) app should get, which could defeat some user-installed malware instances (see the earlier section "Real-World 2FA Attacks").

"OS security improves over time and can mitigate some attack classes."

Moreover, Android introduced SELinux[50] in version 4.3—a security framework that allows more fine-grained access control to system resources. This countermeasure makes it more difficult to perform privilege escalation (also used in our exploits). Further, version 4.3 also introduced authentication for the Android Debug Bridge (adb), which can prevent cross-device infections via USB connections.

The most recent Android version 4.4 provides an enhanced message handling, which prevents third-party applications from silently receiving or sending any SMS. While malware like ZitMo/SpitMo is still able to relay received TAN messages, they will remain visible in the phone's default messaging application, giving the user the chance for an immediate reaction, such as, for example, to call the bank and cancel the transaction. However, this countermeasure will have no effect on our attacks, since we operate at a lower level of the software stack, meaning that the application framework itself will never receive a suppressed message. It is therefore likely that future attacks will follow our concept.

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."

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[51] and Texas Instruments processors have the M-Shield security Extensions.[52] Further, platforms may include embedded Secure Elements (SE) (available, for example, on NFC-enabled devices) or support removable SEs (such as secure memory cards[53] attached to a microSD slot). Finally, SIM cards available on most mobile platforms include a secure element. Such secure hardware allows establishment of a trusted execution environment (TEE) on the device, which can be used to run security-sensitive code to handle authentication secrets in isolation from the rest of the system. Developments in this direction are solutions for mobile payments like Google Wallet[54] and PayPass.[55]

With the release of version 4.3, Android started to support hardware-supported trusted key storage. This means that keys can now be saved in an SE or TEE. However, this is not sufficient to prevent attacks on 2FA schemes, because the keys can be retrieved from the trusted storage by the application that created them. Hence, the adversary could compromise the target application, which has the privileges to query the keys. Even if the OTP generation would take place within the TEE, an attacker could still impersonate the target application in one way or another.

"…the only way to build a secure 2FA on top of TEE is to shift the entire verification process into the TEE."

We believe the only way to build a secure 2FA on top of TEE is to shift the entire verification process into the TEE. We envision the following workflow, which was also described by Rijswijk-Deij[37]: An OTP/TAN application is securely deployed into the TEE. On the first start, this application would establish a secure connection to the service provider/bank (based on public key certificate of the service provider) and prove that it is executed in a legitimate TEE via remote attestation. Next, the application would generate a public/private key pair and send the public key to the service provider/bank. To begin a transaction, the user would start the application. It would then query the service provider/bank for any transaction, which would need to be authorized. If such an action existed, it would be authenticated using the public key of the service provider/bank and displayed to the user, via a trusted user interface. The user would then either allow or deny this action via trusted input. The user's decision would be signed using the generated private key and could be verified by the service provider/bank.

A crucial requirement to underlying TEE in such a use case is trusted user in-/output, which allows the user to enter security sensitive data (such as transaction confirmation) directly into TEE. When such input is mediated by the OS, it can be manipulated by malware so that a program executed within TEE will confirm a transaction or login attempt without user consciousness. However, although some TEEs such as TrustZone can provide trusted user in-/output, in current implementations this feature is not supported. Hence, solutions built on top of existing TEEs still rely on trusted OS components to handle user input.

"…solutions built on top of existing TEEs still rely on trusted OS components to handle user input."

"…most available TEEs are not open to third-party developers."

Moreover, most available TEEs are not open to third-party developers. For instance, secure elements available on SIM cards are controlled by network operators, while processor-based TEEs such as ARM TrustZone and M-Shield are controlled by phone manufacturers. Typically, only larger companies such as Google, Visa, and MasterCard can afford cooperation with phone manufacturers, while smaller service providers remain with an alternative to cooperate with network operators or use freely programmable TEEs such as secure memory cards. However, the solution utilizing SIM-based secure elements would be limited to customers of a particular network operator, while secure memory cards can be used only with devices featuring a microSD slot.

Trusted VPN Proxy
Cross-platform infection attacks as discussed earlier can be defeated by deploying standard countermeasures against MITM attacks. For example, one could enforce HTTPS for every web page request or tunnel HTTP over a remote trusted virtual private network (VPN). However, the former solution would require changes on all Internet servers currently providing HTTP connections (which is infeasible), while the latter would impact performance (as in the case where a single VPN proxy serves several clients). Moreover, it is not clear which party is trustworthy to host such a proxy.

Detection of Suspicious Mobile Apps
SMS-stealing apps exhibit suspicious characteristics or behavior that can be detected by defenders. For example, using static analysis, it is possible to classify suspicious sets of permissions or to identify receivers for events of incoming SMS messages.[56] Similarly, taint tracking helps to detect information leakage.[57] However, tainting requires kernel modifications that are impractical on normal user smartphones and implicit flows can evade taint analysis.[58] An alternative are user space security apps that detect suspicious

behavior of the malicious CitMo/SpitMo/ZitMo apps. Such a security app could, for instance, identify SMS receivers that consume or forward TAN-related SMS by observing the receivers' behavior. Further, by knowing the command and control (C&C) channels of mobile malware, one could identify (and block) data leakage in network traffic.

"…our proposed attack cannot be detected in user space, as we show that we can steal OTPs before any app running in user space has noticed events such as an incoming SMS message."

However, these security measures require prior knowledge of the attacks and C&C obfuscation evades such defenses. Further, our proposed attack cannot be detected in user space, as we show that we can steal OTPs before any app running in user space has noticed events such as an incoming SMS message. Consequently, the aforementioned solutions are not suitable to counter our attack, and instead can only detect the existing SMS-stealing Trojans.

Attack Detection in the Network
Our cross-platform infection attack scenario can be detected or even prevented at the network layer.

"…regular Wi-Fi routers for private use remain unprotected."

Particularly, mitigation techniques exist against rogue DHCP attacks, such as DHCP snooping.[59] For example, the router could stop routing Internet traffic if it detects rogue DHCP servers. However, these mechanisms are available on advanced multilayer switches only and require configuration efforts by network administrators[60], while regular Wi-Fi routers for private use remain unprotected. We did not encounter any home router that uses such countermeasures. Further, these measures are specific to cross-platform infection attacks that rely on rogue DHCP, while ineffective against other scenarios, such as those, for example, based on tethering.

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.[10] 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 that asks the user for permission before performing sensitive operations. Aloul et al.[8,9] 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.[20] 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.[28] propose an authentication technique called QR-TAN that belongs to the class of visual TAN solutions. It requires the user to confirm transactions with the trusted mobile device using visual QR barcodes. Clarke et al.[13] propose to use a trusted mobile device with a camera and OCR as a communication channel to the mobile. The Phoolproof phishing prevention solution[24] utilizes a trusted user cell phone 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 article, an attacker controlling the user's PC can also infiltrate that user's mobile device by mounting a cross-platform infection attack, which undermines the assumption on trustworthiness of the mobile phone.

"…an attacker controlling the user's PC can also infiltrate that user's mobile device by mounting a cross-platform infection attack,…"

Attacks on SMS-based TAN Authentication

Mulliner et al.[21] 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 as dedicated OTP channels that 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.[26] 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.[2] 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 was proof-of-concept malware that had been anonymously sent to the Mobile Antivirus Research Association in 2006.[17][25] 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[22], which spreads via USB keys and targets industrial software and equipment. Further, Wang et al.[32] 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 Wi-Fi access points[4] or ad-hoc peers advertising free public Wi-Fi.[3] When a victim connects to such a network, it gets infected and may start advertising itself as a free public Wi-Fi to spread. In contrast to our scenario, this attack mostly affects Wi-Fi 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 Wi-Fi network. Finally, the infection does not spread across platforms (from PC to mobile or vice versa), but rather affects similar systems.

"…most cross-system attacks were observed in public networks, such as malicious Wi-Fi access points or ad-hoc peers advertising free public Wi-Fi."

Conclusion

"We thus see a need for research on more secure mobile 2FA schemes that can withstand today's sophisticated adversary models."

"As follow-up research, we propose to explore authentication mechanisms that use secure hardware on mobile platforms."

In this article, 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.

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

As follow-up research, we propose to explore authentication mechanisms that use secure hardware on mobile platforms. Although current secure hardware has its limitations (for example, no support for a secure user interface, or not being freely programmable), novel approaches based on secure hardware could eliminate the inherent weaknesses of existing authentication schemes.

References

[1] National vulnerability database version 2.2. http://nvd.nist.gov/.

[2] Kawamoto, Dawn, "Cell phone virus tries leaping to PCs," CNET, http://news.cnet.com/Cell-phone-virus-tries-leaping-to-PCs/2100-7349_3-5876664.html?tag=mncol;txt, 2005.

[3] Phifer, Lisa, "The security risks of 'Free Public WiFi,'" TechTarget, http://searchsecurity.techtarget.com.au/news/2240020802/The-security-risks-of-Free-Public-WiFi, 2009.

[4] Tobadmin, "KARMA demo on the CBS early show," http://blog.trailofbits.com/2010/07/21/karma-demo-on-the-cbs-early-show/, 2010.

[5] Nachreiner, Corey, "Anatomy of an ARP poisoning attack," WatchGuard, http://www.watchguard.com/infocenter/editorial/135324.asp, 2011.

[6] Liebowitz, Matt, "New Spitmo banking Trojan attacks Android users," TechNews Daily, http://www.securitynewsdaily.com/1048-spitmo-banking-trojan-attacks-android-users.html, 2011.

[7] Raiffeisen PhotoTAN. http://www.raiffeisen.ch/web/phototan, 2012.

[8] Aloul, F., S. Zahidi, and W. El-Hajj, "Two factor authentication using mobile phones," in *IEEE/ACS Computer Systems and Applications*, May 2009.

[9] Aloul, F., S. Zahidi, and W. ElHajj, "Multifactor authentication using mobile phones," *International Journal of Mathematics and Computer Science*, 4, 2009.

[10] Balfanz, D. and E. W. Felten, "Hand-held computers can be better smart cards," *USENIX Security Symposium* - Volume 8, USENIX Association, 1999.

[11] Castillo, Carlos, McAfee blog entry: "Android banking Trojans target Italy and Thailand," http://blogs.mcafee.com/mcafee-labs/android-banking-trojans-target-italy-and-thailand/, 2013.

[12] Castillo, Carlos, McAfee blog entry: "Phishing attack replaces Android banking apps with malware," http://blogs.mcafee.com/mcafee-labs/phishing-attack-replaces-android-banking-apps-with-malware, 2013.

[13] Clarke, D., B. Gassend, T. Kotwal, M. Burnside, M. v. Dijk, S. Devadas, and R. Rivest, "The untrusted computer problem and camera-based authentication," in *International Conference on Pervasive Computing*, Springer-Verlag, 2002.

[14] 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.

[15] Cronto Limited, "CorpBanca and Cronto secure online banking transactions with CrontoSign, http://www.cronto.com/corpbanca-cronto-secure-online-banking-transactions-crontosign.htm, 2011.

[16] Malik, Amit, "DLL injection and hooking," SecurityXploded, http://securityxploded.com/dll-injection-and-hooking.php

[17] Evers, J., "Virus makes leap from PC to PDA," CNET, http://news.cnet.com/2100-1029_3-6044457.html, 2006.

[18] Jerschow, Y. I., 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.

[19] Kalige, E. and D. Burkey, "Eurograbber: How 36 million euros was stolen via malware," http://www.cs.stevens.edu/~spock/Eurograbber_White_Paper.pdf.

[20] Mannan, M. and P. C. Van Oorschot, "Using a personal device to strengthen password authentication from an untrusted computer," in *FC'07/USEC'07*, 2007.

[21] Mulliner, C., R. Borgaonkar, P. Stewin, and J.-P. Seifert, "SMS-based one-time passwords: Attacks and defense" (short paper), in *DIMVA*, July 2013.

[22] Falliere, N., "Exploring Stuxnet's PLC infection process," Symantec blog entry, http://www.symantec.com/connect/blogs/exploring-stuxnet-s-plc-infection-process, 2010.

[23] V. News, "Teamwork: How the ZitMo Trojan bypasses online banking security," Kaspersky Lab, http://www.kaspersky.com/about/news/virus/2011/Teamwork_How_the_ZitMo_Trojan_Bypasses_Online_Banking_Security, 2011.

[24] Parno, B., C. Kuo, and A. Perrig, "Phoolproof phishing prevention, in *Financial Cryptography and Data Security*, Springer-Verlag, 2006.

[25] Peikari, C., "Analyzing the crossover virus: The first PC to Windows handheld cross-infector," *InformIT,* http://www.informit.com/articles/article.aspx?p=458169, 2006.

[26] Schartner, P. and S. Bürger, "Attacking mTAN-applications like e-banking and mobile signatures," Technical report, University of Klagenfurt, 2011.

[27] Sparkasse Pfullendorf-Meßkirch, "Online banking mit chipTAN," https://www.sparkasse-pm.de/privatkunden/banking/chiptan/vorteile/index.php?n=/privatkunden/banking/chiptan/vorteile/.

[28] Starnberger, G., L. Froihofer, and K. Goeschka. "QR-TAN: Secure mobile transaction authentication," in *International Conference on Availability, Reliability and Security*, IEEE, 2009.

[29] Tanenbaum, A., "*Modern Operating Systems*", 3rd edition, Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

[30] 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.

[31] van der Veen, V., N. dutt Sharma, L. Cavallaro, and H. Bos, "Memory errors: The past, the present, and the future," in *Recent Advances in Intrusion Detection Symposium*, 2012.

[32] Wang, Z. and A. Stavrou, "Exploiting smart-phone USB connectivity for fun and profit," in *26th Annual Computer Security Applications Conference*, ACM, 2010.

[33] Zhou, Y. and X. Jiang, "Dissecting Android malware: Characterization and evolution, in *IEEE Symposium on Security and Privacy*, 2012.

[34] "Mobile Malware Mini Dump," Contagio Mobile, http://contagiominidump.blogspot.de/

[35] Lee, Byoungyoung, Long Lu, Tielei Wang, Taesoo Kim, and Wenke Lee, "From Zygote to Morula: Fortifying Weakened ASLR on Android," in *IEEE Symposium on Security and Privacy*, 2014.

[36] Roland van Rijswijk-Deij, Roland and Erik Poll, "Using trusted execution environments in two-factor authentication: comparing approaches," in Open Identity Summit 2013 (OID-2013), volume 223 of Lecture Notes in Informatics.

[37] PaX Team, http://pax.grsecurity.net/

[38] Shacham, Hovav, "The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86)," in *14th ACM conference on Computer and Communications Security*, CCS '07. ACM, 2007

[39] Buchanan, Erik, Ryan Roemer, Hovav Shacham, and Stefan Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *CCS' 08: Proceedings of the 15th ACM Conference on Computer and Communications Security*, ACM, 2008

[40] PaX Team, PaX address space layout randomization (ASLR), http://pax.grsecurity.net/docs/aslr.txt

[41] Shacham, Hovav, Eu Jin Goh, Nagendra Modadugu, Ben Pfaff, and Dan Boneh, "On the effectiveness of address-space randomization," in *CCS' 04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, ACM Press, 2004.

[42] Bhatkar, Sandeep, R. Sekar, and Daniel C. DuVarney, "Efficient techniques for comprehensive protection from memory error exploits," in *USENIX Security Symposium*, 2005.

[43] Kil, Chongkyung, Jinsuk Jun, Christopher Bookholt, Jun Xu, and Peng Ning, "Address space layout permutation (ASLP): Towards fine-grained randomization of commodity software," in *Annual Computer Security Applications Conference*, 2006.

[44] Pappas, Vasilis, Michalis Polychronakis, and Angelos D. Keromytis, "Smashing the gadgets: Hindering return-oriented programming using in-place code randomization," in *IEEE Symposium on Security and Privacy*, 2012.

[45] Hiser, Jason D., Anh Nguyen-Tuong, Michele Co, Matthew Hall, and Jack W. Davidson, "ILR: Where'd my gadgets go?" in *IEEE Symposium on Security and Privacy*, 2012.

[46] Wartell, Richard, Vishwath Mohan, Kevin W. Hamlen, and Zhiqiang Lin, "Binary stirring: Self-randomizing instruction addresses of legacy x86 binary code," in *ACM Conference on Computer and Communications Security*, 2012.

[47] Giuffrida, Cristiano, Anton Kuijsten, and Andrew S. Tanenbaum, "Enhanced operating system security through efficient and fine-grained address space randomization," in *USENIX Security Symposium*, 2012.

[48] Snow, Kevin Z., Lucas Davi, Alexandra Dmitrienko, Christopher Liebchen, Fabian Monrose, and Ahmad-Reza Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *34th IEEE Symposium on Security and Privacy*, 2013.

[49] Bittau, Andrea, Adam Belay, Ali Mashtizadeh, David Mazieres, and Dan Boneh, "Hacking blind," in *35th IEEE Symposium on Security and Privacy*, 2014.

[50] National Security Agency, Security-Enhanced Linux, http://www.nsa.gov/research/selinux.

[51] Alves, Tiago and Don Felton, "TrustZone: Integrated hardware and software security," *Information Quarterly*, 3(4), 2004.

[52] Azema, Jerome and Gilles Fayad, "M-Shield mobile security technology: Making wireless secure," Texas Instruments white paper, 2008.

[53] Giesecke & Devrient Press Release, "G&D Makes Mobile Terminal Devices Even More Secure with New Version of Smart Card in MicroSD Format," http://www.gi-de.com/en/about_g_d/press/press_releases/G%26D-Makes-Mobile-Terminal-Devices-Secure-with-New-MicroSD%E2%84%A2-Card-g3592.jsp.

[54] Google Wallet, http://www.google.com/wallet/how-it-works/index.html.

[55] MasterCard Contactless, "Tap to pay," http://www.mastercard.us/paypass.html#/home/,2012.

[56] Zhou, Yajin, Zhi Wang, Wu Zhou, and Xuxian Jiang, "Hey, you, get off of my market: Detecting malicious apps in official and alternative Android markets," in *19th Annual Network and Distributed System Security Symposium*, 2012.

[57] Enck, William, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[58] King, Dave, Boniface Hicks, Michael Hicks, and Trent Jaeger, "Implicit flows: Can't live with Em, can't live without Em," on *Information Systems Security*, Springer, 2008.

[59] Bhaiji, Yusuf, "Understanding, preventing, and defending against layer 2 attacks," Cisco, http://www.nanog.org/meetings/nanog42/presentations/Bhaiji_Layer_2_Attacks.pdf.

[60] Cisco, "Configuring DHCP Snooping," http://www.cisco.com/c/en/us/td/docs/switches/datacenter/sw/4_1/nx-os/security/configuration/guide/sec_nx-os-cfg/sec_dhcpsnoop.pdf.

Author Biographies

Alexandra Dmitrienko is a research assistant at Fraunhofer Institute for Secure Information Technology in Darmstadt (Germany). She obtained her MSc in IT-Security from the Saint Petersburg State Polytechnical University in Russia in 2007. Her academic achievements were honored by the Intel Doctoral Student Honor Award in 2013. Her research is focused on mobile operating system security, runtime attacks, and secure mobile applications (online banking, access control, mobile payments and ticketing). She can be contacted at alexandra.dmitrienko@sit.fraunhofer.de.

Christopher Liebchen is a student at the Technical University of Darmstadt and currently working on his master's degree in IT-Security. His research focuses on system/mobile security, reverse engineering, and runtime attacks. Results of his work have been presented at scientific as well as at industrial conferences. He can be contacted at christopher.liebchen@cased.de.

Christian Rossow is a postdoctoral researcher at the Vrije Universiteit Amsterdam (The Netherlands) and at the Ruhr University Bochum (Germany). Christian completed his PhD studies at the VU Amsterdam in April 2013, and his PhD dissertation was awarded with the prestigious Symantec Research Labs Fellowship award in 2013. His research interests are malware analysis, mobile security, botnet tracking, and denial-of-service attacks. He can be reached at c.rossow@vu.nl.

Ahmad-Reza Sadeghi is a professor of computer science at Technische Universität Darmstadt, Germany and the head of the System Security Lab at the Center for Advanced Security Research Darmstadt (CASED). Since January 2012 he has been the Director of the Intel Collaborative Research Institute for Secure Computing (ICRI-SC) at TU-Darmstadt. He holds a PhD in computer science from the University of Saarland in Saarbrücken, Germany. Prior to academia, he worked in research and development of telecommunications enterprises, such as Ericsson Telecommunications. He is on the Editorial Board of the ACM Transactions on Information and System Security. He can be reached at ahmad.sadeghi@trust.cased.de.