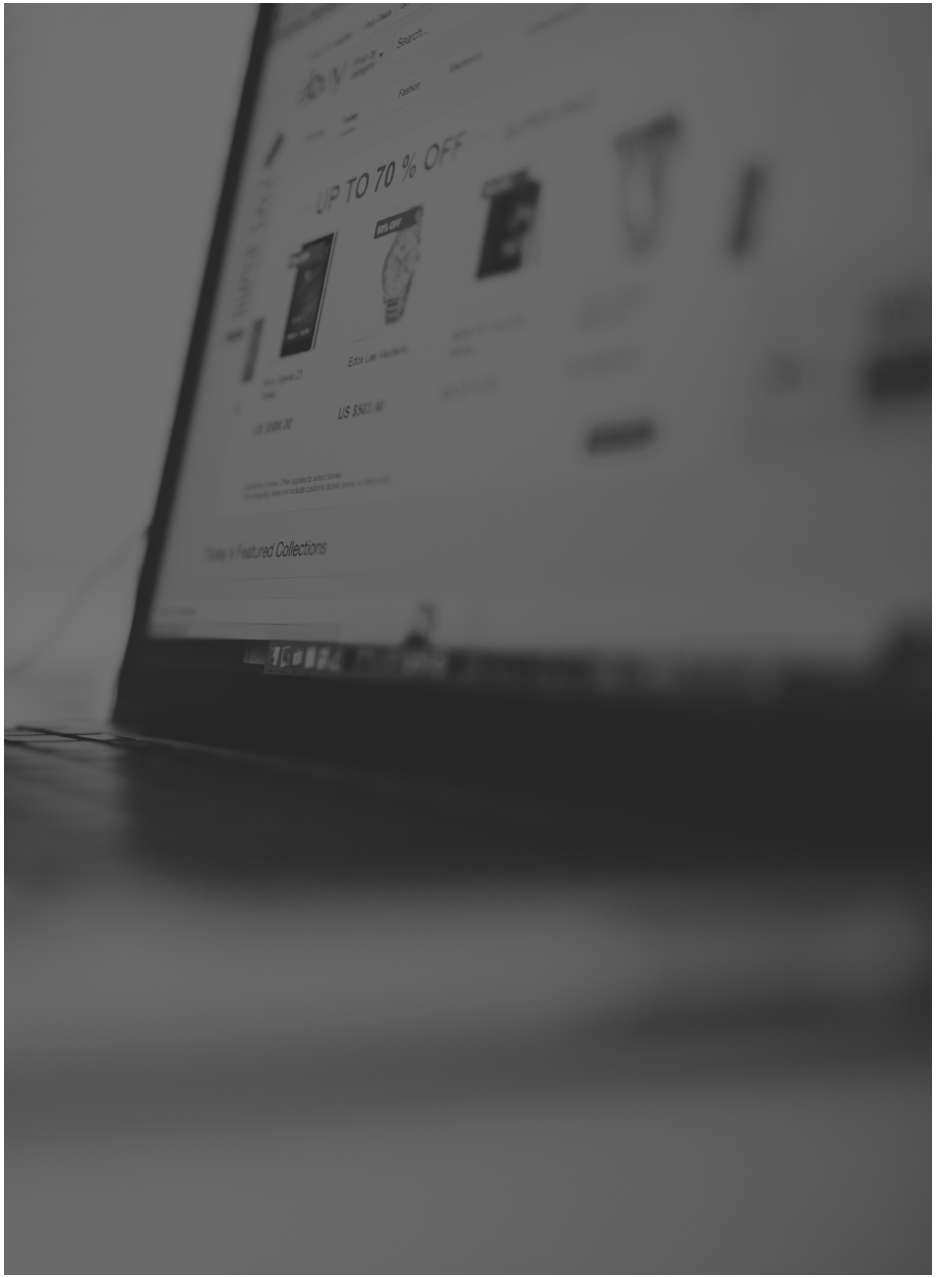
The Level of Security of Mobile Banking Applications in Poland

NOVEMBER 2016

PGS SOFTWARE

INTRODUCTION



The question of whether a given banking application (or an 'app') is safe can be understood in several ways. The most obvious interpretation concerns the security of financial resources. However, if we think about it for a little while, we should also take into account, for example, the security of personal data and transaction history.

In this report, we review nineteen mobile apps for Android published by retail banks operating in Poland. We opened the most basic account with each bank, activated electronic and mobile banking services, installed the relevant application, and performed a few simple operations. At the same time, we attempted to intercept transmitted data, familiarised ourselves with the structure of the app and information stored by it, and examined good and bad practices that were followed while developing the software.

It needs to be stressed that we are not making a safety ranking here or saying that app A is better than app B. Why? We've only spent a few hours on each app, which means that we almost certainly overlooked a significant share of faults and vulnerabilities. Reliable penetration tests would have required much more time.

We tried to keep this publication understandable for the average smartphone user, and hence, we've explained the more difficult terms in boxes.

We would like to apologise in advance to some banks for deviating from using their full business and brand names. After a series of mergers and acquisitions, they tend to be quite long and not very convenient to use in certain cases, so we took the liberty to go with their common names.

I. APP TESTS

Conclusions

The quality and security of many of the apps examined leaves a lot to be desired. During our short tests, we managed, among others, to take over the session of a logged-in user, to cause a data breach, and to intercept a section of a masked password. We also found log-in IDs, session tokens, as well as, data on a number of specific services in the system logs.

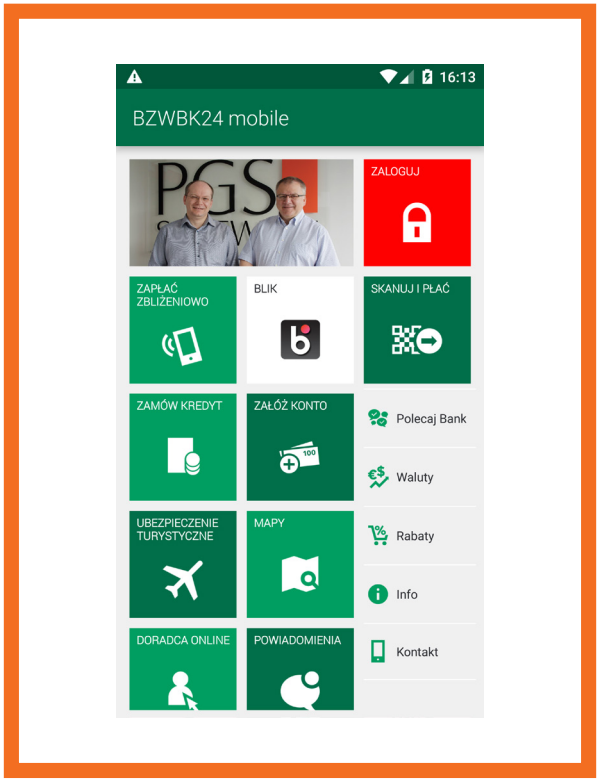
We intercepted data transmitted by many of the apps. In some cases, data was exchanged in an open-text format, which means it could be modified at any point of its transmission between the sender and the recipient. Other apps did not seem to mind fraudulent security certificates, which we used purposely to allow decryption of the data transmitted.

Shockingly, we discovered that in many cases apps submitted to the store did not undergo any quality control procedures. This is a cause for great concern. How then are we supposed to believe that an app whose resources contain a set of test log-in details for a working environment has passed any tests? Or an app with malfunctioning maps? Or with mandatory updates that can be skipped with just the tap of a finger?

From our observations, we believe that the root cause lies in the inadequate quality of the programming for financial apps. Because even though a piece of code that allows automatic form filling or server address changes may be necessary during the test phase, according to good computing practices, it should not be included in the Appstore version.

The development of parts of the apps under examination was outsourced to external companies. This indicates that the parts were not seen as key components of the infrastructure, and that a bank may not be competent to assess the quality of the end product received. Moreover, banks that treat mobile apps neglectfully will find it difficult to employ experts in this technology, which further reinforces the adverse practices.

The future of mobile banking remains unwritten. So far, convenient access to accounts and cashless transactions (such as the Polish payment service **BLIK** and virtual payment cards) are the priority. But we can imagine a scenario in which the theft of funds with the use of tens or hundreds of thousands of phones would result in a complete departure from mobile banking. Banks should be interested in making sure that this remains only a fictional scenario.



This is what the start menu of the **BZ WBK** app could look like if we used it within the range of the PGS Software network.

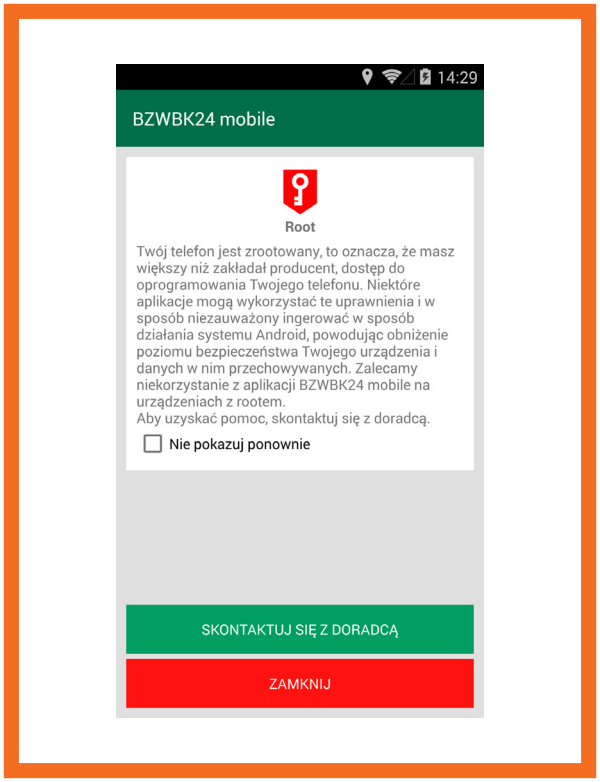
Android and security

Before we proceed to analysing specific cases, here's a short note on Android. Hundreds of devices running on this system, offered by tens of different manufacturers, are now available on the market. A large part of them is sold by mobile network operators who pre-install their own software, which means that the number of possible software configurations is in the thousands.

Meanwhile, software updates, which patch up any loopholes in the operating system, are generally only available for the more expensive models, and only for a year or two after they have been released on the market. As a result, some estimates[1] suggest that as much as 80% of Android devices are vulnerable to external attacks . In some cases, the user needs to install a malicious application or visit a website to put themselves at risk. However, in the case of a vulnerability known as ***Stagefright***[2], the attacker may activate any code by sending an MMS message with specially crafted contents, as a result of which the malicious code is immediately activated.

Significant mistakes can also be found in specific models or device series. In 2015, the vulnerability of Samsung S4, S5 and S6 devices to attacks made with the use of a fraudulent keyboard software update was announced – also, in this case, the use of a malicious code was possible without the end user even knowing that this was happening. All that was needed was an active internet connection[3].

At the time of writing this article, there was a lot of talk about the ***Rowhammer*** (memory contents modification) and ***Dirty COW*** (increased process privileges) vulnerabilities; each of them may be used within a programme that does not require any special permissions.



The **BZ WBK** app displays a warning about having a rooted device.

On enemy soil

Banking app developers must think about how the app is going to work on a compromised device. For example, blocking the screenshot function may make it impossible to send a picture of a completed bank transfer, but it also prevents PIN code interception. The use of certificate pinning, although it may seem excessive in theory, will prevent data interception in the event that a fraudulent certificate has been injected in the system by the attacker. We will talk more about that in the following chapters.

It may be assumed that relatively often malware can read logs and modify work files created by apps. This means that user data should be stored in an encrypted form, and static data should be protected with a checksum or a digital signature to detect unauthorised modifications. The scale of such attacks may be very large.

The modification of the banking app itself by the attacker would be more difficult. We should rather be aware of programme overlays which display data-phishing forms over the actual, original app – such attacks are already taking place. Network traffic interception, also difficult, seems more likely in the case of attacks directed at a specific user, company, or network. We would not expect large-scale attacks here, but smaller attacks may be more damaging.

The current situation in Poland

If it is so bad, why have we not heard about large-scale attacks on mobile banking services in Poland yet? There are a few reasons for this. First of all, several dozens of different variants of offensive code would be required for the thousands of software configurations found on the market, and they may not be tested sequentially (because, for example, the first failed attempt would make the phone freeze). Secondly, there are at least around a dozen banking apps to be attacked, which means that it is necessary to look for vulnerabilities in each of them individually. Thirdly, although malware may have easy access to files and system logs, intercepting data from the memory of an active app or intercepting network communications is very difficult (even more so that such an attack must be carried out autonomously). Fourthly, the mobile channel often does not allow savings account closure or loan application submission, which means that the thief can only take out funds from the current account.

1. http://androidvulnerabilities.org/
2. https://www.kb.cert.org/vuls/id/924951
3. https://www.nowsecure.com/blog/2015/06/16/remote-code-execution-as-system-user-on-samsung-phones/

This means that while malware developers achieve the highest return on creating global botnets for hire or encrypting data and collecting ransom (***ransomware***), mobile banking services in a medium-sized European country will not be the prime target. Especially if an attack attempt would be made from abroad, because it would require the use of text written in perfect Polish.

This does not mean, however, that banks should sleep easy. This brings to mind an old joke – ***while running from a lion you don't have to be first; all you need is to not be last***. Analogically, without constant improvements, an app will fall to the bottom of the ranking – as the most vulnerable one – and attract the majority of attacks. A serious new-generation attack will use all the functions that marketing specialists are so happy about today. For example, in the world of geolocation and consumer behaviour analysis it will not be necessary to create a fake account. However, simply buying a cinema ticket and switching your phone off will be enough. By doing so, you will create a situation in which your account will be able to be used for two hours as an interchange station for thieves to make a series of quick transfers, which, if you are lucky, you will find out about from text messages received after the film.

While we are on the subject of text messages – we can safely assume that malware is able to read, mute, and forward text messages with authorisation codes to any third party. That is why it is entirely possible that in the future we may go back to using paper lists of one-off codes.

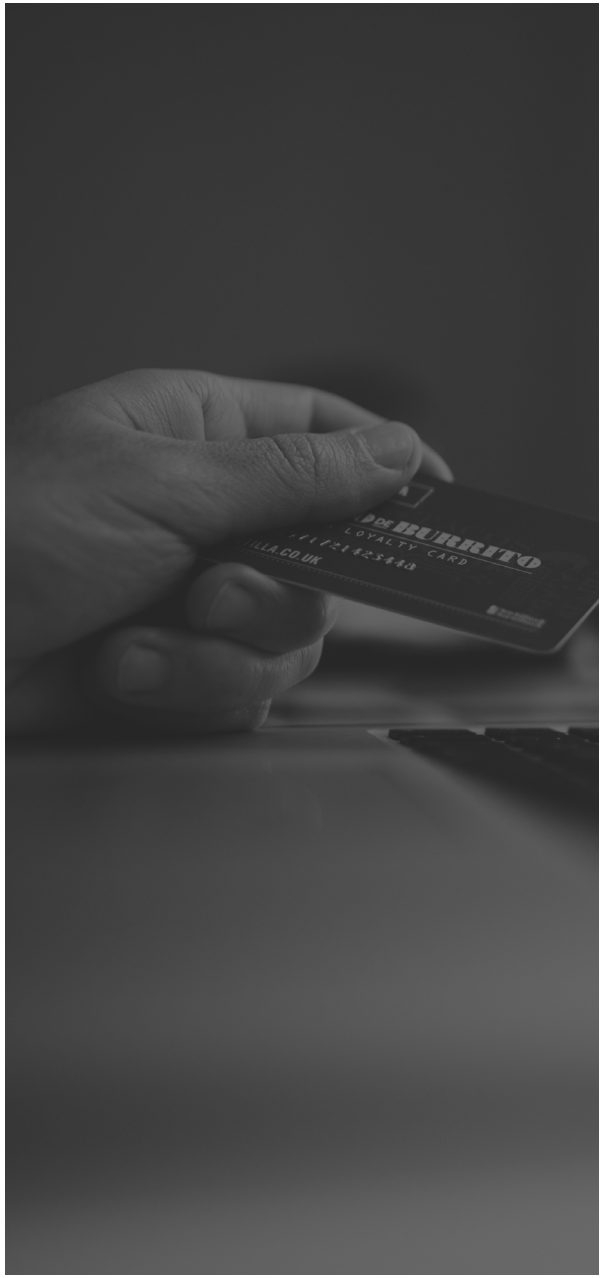
Tested app versions

The tested versions of the apps were those available on 1 July 2016 and the current versions at the time of testing (between July and October). Descriptions of all the mistakes and vulnerabilities found were sent to the relevant banks asking them to correct the mistakes and informing them that the issues found will be mentioned in this report.

Not all faults have been corrected. Some of them, although we are writing about them in past tense, are still present as we finish preparing this article.

Critical vulnerabilities

Critical vulnerabilities are the ones that may lead to a loss of funds or data leakage on a ***healthy*** device. During our tests, we found at least several of them. They were also accompanied by two mistakes, which clearly show the banks' disregard for good practice.



BZ WBK

personal data leakage

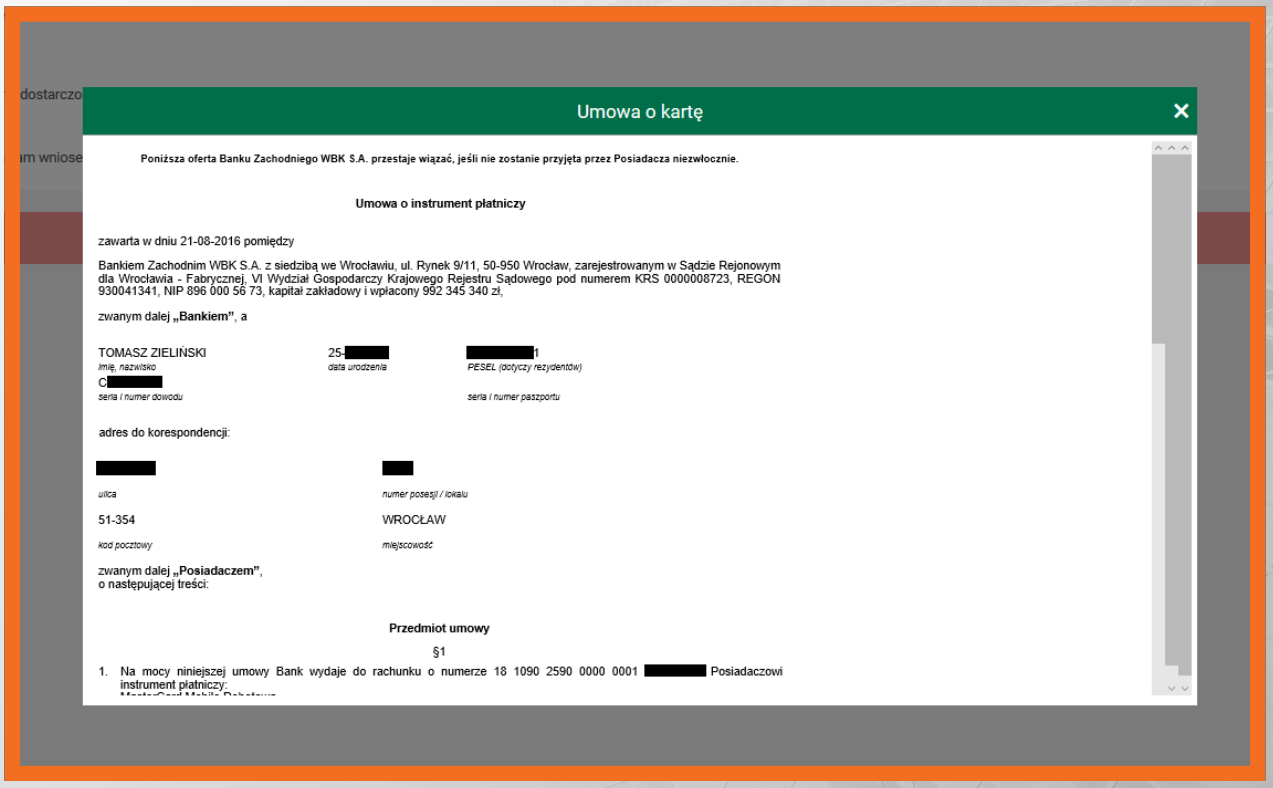
If a user has no payment card and chooses the contactless payment option, it triggers a new card application form. The form is not, however, a part of the programme, instead it is displayed on a web browser embedded in the app. The problem is that the address of the form, containing a one-off token and user ID, is recorded in the logs from where it could be stolen by attackers. As a result, it was possible to order an unwanted service in the user's name and to gain access to their personal data (first and last name, address, personal ID number "PESEL", and ID card number) displayed in the form.

What is worse, the form – despite us having already logged out of the mobile app – remained active for over 30 minutes, which allowed for a delayed cyberattack to be made. It was possible to open the form on a PC, operating in a completely different network than the mobile phone (such a change should be detected and treated as an attack).

On a technical note – system logs

Logcat system logs in Android are data streams, where applications can leave their entries. The intended purpose of the logs are diagnostics – entries about all incidents and programme status updates can be sent to the logs, they may also contain information about the causes of application malfunctions. But information that is helpful to a programmer may also provide an attacker with some valuable clues. In line with good practice, published versions of apps should not use this mechanism at all.

In Android, up to version 4.0, applications with the right permissions could intercept entire data streams. In version 4.1 and later ones, that possibility was removed. The log-in system buffer stores several thousands of recent items, so it is possible to find entries made hours earlier.



A form intercepted from a mobile phone can be opened on a PC.

ING

session hijacking

A critical error in the **ING** app was also caused by the leakage of links to system logs, but in this case, the culprit was... the security mechanism of a transaction service protecting it against self-XSS attacks. With this security mechanism, the website displayed a warning in the JavaScript console, advising the user not to paste any contents in the window. In Android, the browser has no console, which means that the alert would be sent to the system logs, while the browser added the source page address to the alert.

But where did the link come from? The older version of the mobile app (renamed during the tests to *ING dla Przedsiębiorców*) included a button which, at the click of a finger, redirected a logged-in user to the web application without the need to log in again (so-called Single Sign-On, SSO). That link, generated by the app and sent to the mobile browser, would be sent to the logs as described above.

It turned out that when we immediately sent the address found in the logs to a computer and opened it in several browser windows, the PC would win that race and the logged-in user's session was able to be hijacked from the mobile, which gave the attacker access to the victim's transactional platform.

The warning displayed in the browser console was the cause of the issues on the mobile device.

On a technical note - Self-XSS

An XSS error means that an undesired code injected by the attacker has been executed by the browser. The result may be, for instance, purchasing goods from an online auction site without any involvement of the user. XSS attacks on Facebook were quite a common occurrence, with the victims automatically publishing content on their own wall or their friends' walls (infecting them at the same time).

Self-XSS attacks spread with the use of socio-technical methods – the victim needs to be persuaded to visit a specific website, open the JavaScript console (only useful for programmers), and paste a piece of code in the console window. Facebook users have been tempted, for example, with the ability to view private messages or to intercept another user's account, but as a result, it was them that became victims of a cyber-attack.

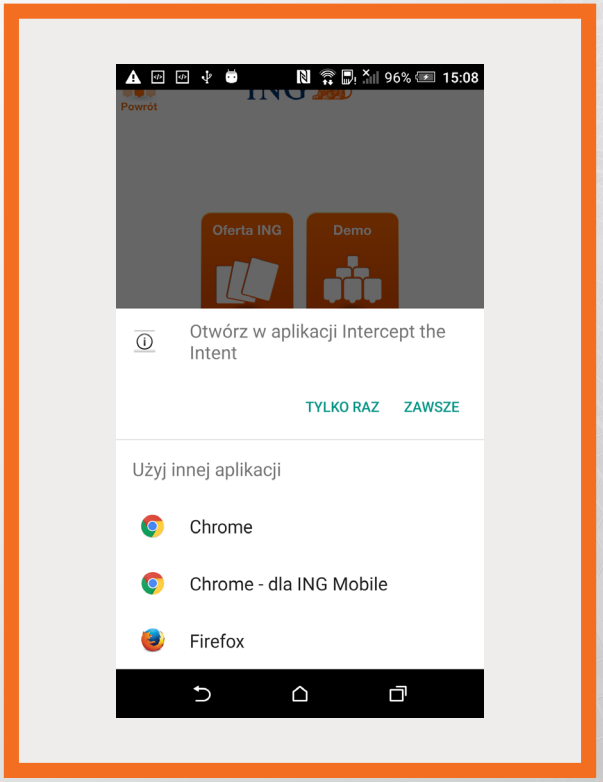
ING

The use of implicit intents allows interception of SSO links

The mechanism which allows opening a logged-in session in a browser window, as described above, causes another issue as well – the application used the system *intents* mechanism, i.e. the ability to choose which programme is to carry out a triggered action.

The mechanism itself gives Android a big competitive advantage. It allows users to transfer data between apps freely and programmers to define which types of actions and data a given app should be registered for. In consequence, a *Share Text* action will trigger a list of suggestions to send the text to/via a text message, e-mail, Skype, Evernote, and other programmes that can use text data.

An issue may arise when we want to share a secret (here: SSO web address) in this way. The attacker may register for the *View HTTPS* action, entering links to the *ingbank.pl* website as a filter. Their action may then be visible in the menu, for example as *Chrome for ING*. If the user chooses that option, the session will be hijacked or – what is much worse – the contents will be substituted in such a way as to make the user believe, for as long as possible, that they are using a normal web application.



Chrome for ING is a fake, planted by us – this time.

PEKAO

masked password leak

The **Pekao** app sends copies of all the network communications to the logs (bad practice). During our communication exchange, individual letters of a masked password were transferred in the open text format (bad practice; the result of the one-way hash function, i.e. hash, should be transmitted). The app did not block communications with the use of a self-trusted security certificate either (bad practice).

So, if you use your phone at work and you had to install the company certificate in order to use the Wi-Fi, make sure you stay on good terms with the admin. After logging into the **Pekao** app a few times, they will know your full password.

Pekao application – a copy of the network traffic got into the system logs.

IDEA BANK

sending full personal data to the app

We will come back to the technical aspects of the transmission channel security later. Let's just mention that the **IdeaBank** app did not prevent data interception by means of using of a proxy and a fraudulent certificate, which for a mobile financial application is a breach of good practice.

When we look into data transmitted from the server to the mobile app, it may send chills down our spine. Apart from the first and last name, address, and PESEL (personal identification) number, we can find our phone number, mother's maiden name, and the ID card number, (as well as, data processed internally by the bank; for example, the person writing these words had been classed as an *higher-risk* client).

As a general rule, information which is not processed by the mobile app should not be shared with the app in the first place. A successful interception of data from the **IdeaBank** app at most telephone customer service centres is sufficient to reset the password and intercept all of the log-in data. This means that it may be the first step of an attack on a specific individual, performed across a number of institutions.

```
rozne okresy w lokacie;47507073;Solaris86;1005871
57178520;57178520;123456Qq;none
lokatowywow;lokatowywow;123456Qq;1006962
49884385;49884385;Solaris86;none
Synchro1;Synchro1;123456Qq;none
22718740;22718740;Solaris86;none
Solarissss;Solarissss;Solaris86;none
edycja blad;36620706;123456Qq;none
user1;17345296;Marcin12;2964
user1a;86704303;Marcin12;1000565
user2;52160468;Marcin12;1000568
user3;56436841;123456Qq;1002921
user4;27251376;123456Qq;1002942
user5;31274110;123456Qq;1002922
user6;12083370;5169389543;1003053
user8;24843014;Przemek1;1002307
user9;44281517;123456Qq;1003462
user10;70045088;Marcin12;1000897
user12;10773883;123456Qq;1003793
user13;96659482;Marcin12;5593
user14;35767912;Kanapka6;1001703
user15;66723493;Przemek1;1002299
user16;97366064;123456Qq;1005394
user17;43279804;Marta123;1003822
user18;88309903;Goosip2008;7491
user19;60207105;123456Qq;1005442
user20;46626671;123456Qq;1005598
user21;97404421;123456Qq;1005603
AnnaMaria;AnnaMaria;Marta123;none
Historia;17261962;123456Qq;none
Struktury;47403899;Marta123;1008314
Lokata STX;45072643;123456Qq;1010763
TylkoLokata;94972961;123456Qq;none
BezKont;94603329;123456Qq;none
BezKont;93401487;123456Qq;none
BezKont;92549661;123456Qq;none
BezKontPerson;38580461;123456Qq;none
BezKontFirma;85607457;123456Qq;none
PersonOnly;50093075;Solaris86;none
WnioskiLokaty;74805641;Solaris86;none
user28532;123456Qq;123456Qq;1005966
user28530;71453469;123456Qq;1005965
user28528;66994323;123456Qq;1005964
```

Logins and passwords found in the **IdeaBank** app.

Test account passwords in the app's resources

The resources of the **IdeaBank** app contained a text file with over a hundred test log-ins and passwords. Some of them worked in a production environment. We found out that the most popular password among the testers was "123456Qq", a few steps ahead of "Kanapka6" and "Solaris86". Now, we would like to forget about that as soon as possible.

Communications security

On a technical note

How do we know if a server connection is secure? Online banking users have got used to checking the green padlock symbol in the address bar – if it is present, that means that the browser is certainly connecting to the website of a given bank and that the connection is secure. From a technical perspective, while establishing a connection, the unforgeable public key, signed by one of the trusted certification authorities, is verified.

Every computer and mobile phone contains a list of several dozens to several hundred trusted certification bodies. An issue arises if an attacker manages to add themselves to that list (all they need is a minute alone with the keyboard). From then on, they can issue a similar certificate, forge the green padlock and intercept and modify the entire incoming and outgoing network traffic. That is why banks keep reminding us that no plug-ins, add-ons, "safer" certificates, "additional security mechanisms" or similar solutions need to be installed to use mobile banking in a secure way.

If the list of trusted certification bodies is "infiltrated" by the attacker, there is one last line of defence. App developers can embed a fixed certificate in the app to make it possible to check that we are connecting to the target server without any undesired "middlemen". This is called certificate pinning. It is part of good practice, which should be followed by all financial application developers.

Mobile apps must communicate with bank servers. That is where they get information about active services, account history, balance, and where they send transfer orders, bond opening instructions, and credit card applications. An attacker will want to intercept such information (account history and transactions may contain confidential data) and to modify it (for example, in order to swap the account number for outgoing transfers). Therefore, communication mechanisms must ensure both confidentiality and integrity of messages transmitted.

Did the applications tested meet our expectations in this regard? Sadly, not entirely.

Unimplemented good practices

During the tests it turned out that not all applications meet the expected standards. Many gave up without a fight once the network traffic was streamed from the mobile to a PC and a proxy server with its own security certificate was launched.

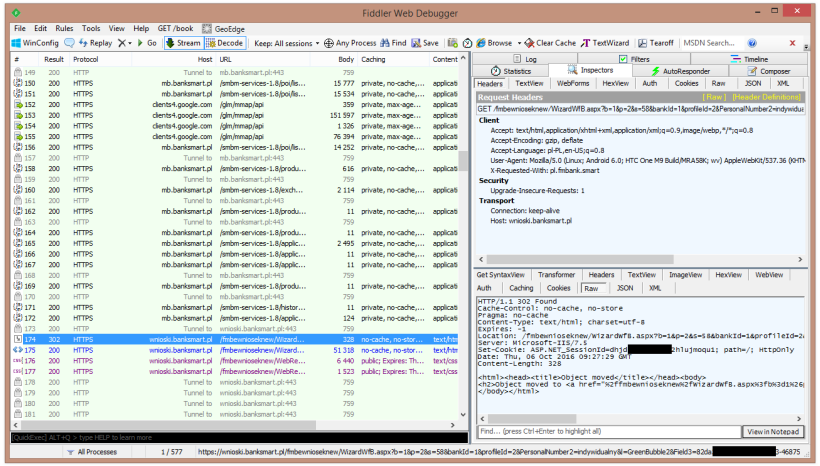
In the case of the **mBank**, **IdeaBank**, **T-Mobile Usługi Bankowe**, **Orange Finanse** and **Bank Smart** apps – the transmission channel was wide open to us. A traffic-intercepting proxy made it possible to modify the requests and responses. As mentioned before, in the case of **IdeaBank**, just intercepting data may have devastating consequences.

The software published by **Pekao**, **Bank Zachodni**, **Raiffeisen**, and **Citi** banks also communicates with the server part, which makes it possible to intercept additionally encrypted data and data secured with a digital signature. This enables cybercriminals to systematically search for vulnerabilities in the security mechanisms of intercepted communications, even if it is not possible to modify data freely.

Another issue concerns applications based on web technologies, which download executive modules from the Internet (such as **Citi** or the new version of the **ING** app). In such a case, a single injection of a hostile code is enough to permanently compromise all other security measures.

The **Getin**, **Millennium**, **BGŻ**, **BPH**, **Alior Bank**, **Eurobank** and **Credit Agricole** apps block communications completely. **Millennium Bank** deserves a special mention here, as a moment after preventing a log-in attempt the app sends a copy of the fraudulent certificate to the head office. We are guessing that our substitute certificate named DO_NOT_TRUST_ FIDDLER_ CERTIFICATE (proxy server certificate) did not really raise an alarm, but the reporting mechanism described can generate a timely warning about an attack being launched.

The **Millennium** app, praised a moment ago, must, however, be reprimanded for saving the history of the user's operations in a file and for storing such files over prolonged periods of time. The app catalogue also featured a cache with files downloaded from the Internet, which had been sitting there for months.



Communications with bank servers can be intercepted, for example, with the use of the Fiddler software.

No data transfer encryption

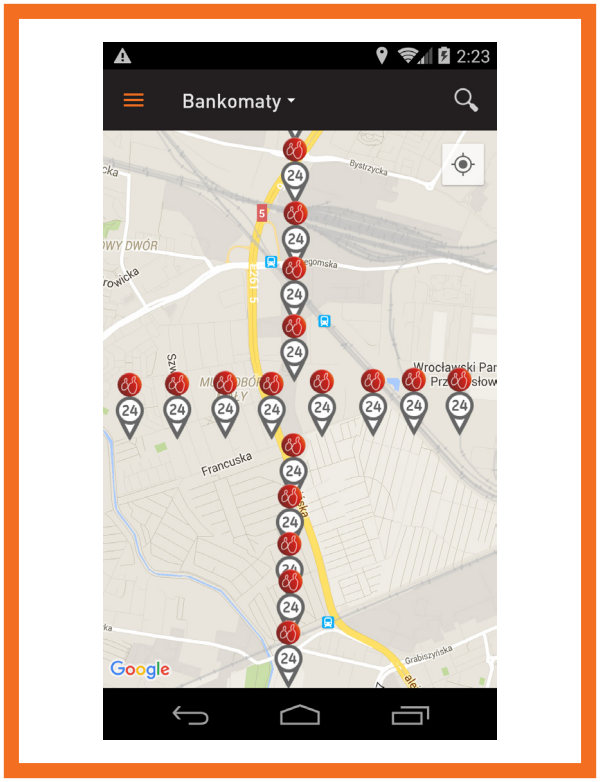
Not all Internet users are aware that ***standard*** network connections (for example, established with the popular HTTP protocol) can be intercepted and modified on all devices involved in the transmission of data. When the first apartment block and community computer networks started to appear in Poland, with the high costs of a fixed broadband connection shared between the network members, self-taught administrators would play jokes on them, for example, by horizontally flipping all of the pictures on websites visited by a given user. The more intrusive administrators would intercept and read the network members' e-mails.

Today, Internet access is provided by professional Internet providers who do not care for such simple jokes, however, the risk of a *Man in the Middle* attack (which involves the falsification of a query or answer) is still real – especially if we sometimes use wireless city or hotel networks.

That is another area in which the mobile banking application programme developers let us down – we found links to resources accessible through an unsecured HTTP protocol in the apps of **PKO BP**, **Pekao**, **BZ WBK**, **mBank**, **ING**, **Getin Bank**, **BGŻ**, **BPH**, **Alior Bank**, **IdeaBank**, Eurobank and **CreditAgricole**. Sometimes they were links to PDF documents, other times to terms and conditions, websites listing the services offered, and even links from the main application menu.

However, there were more spectacular failures – the **BZ WBK** app downloaded an image to the main menu via a non-encrypted connection. The possible consequences of this are outlined on page 3 of this report. The **BPH** app downloads a list of cash machines in this manner – an injected list may send **BPH** customers to machines operated by the bank's competitors. Injected foreign exchange rates can cause short-lived enthusiasm, or a heart attack. The mobile application's safety instructions loaded via the unsecured HTTP connection are quite pitiful (**Alior Bank**, **BZ WBK**, **PKO BP**).

Why is the lack of encryption dangerous? Let's imagine that an attacker intercepts a request to download terms and conditions applicable to fees and charges, and sends the user a document with a message that reads: "Untrusted device. The document can be downloaded after an extended security certificate has been installed. Do it now! Download the certificate from[...]". The million dollar question is – how many people will unwittingly install malware, believing that they are following the bank's instructions?



Map of cash machines downloaded through an unsecured HTTP connection.

Problems with bank websites

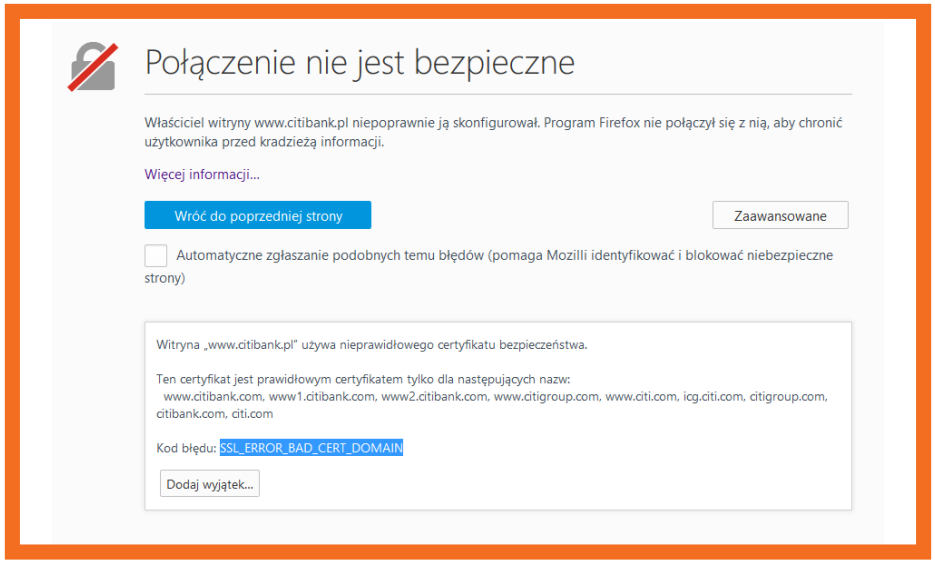
While on the topic of network communications, we must also criticise banking practices related to the websites of several of the banks examined here.

A moment ago we were talking about the dangers associated with transmitting data through an unsecured HTTP protocol. It may be hard to believe but there are still banks on the market today whose websites are not secured with an encrypted connection. If the user manually adds *https* to the address of the **Bank Smart** website, they will be transferred to an untrusted http connection. The same goes for inteligo.pl and iko.pkobp.pl, from the **PKO BP** brand. In the summer of 2016, we achieved the same result on the pkobp.pl and pekao.com.pl websites; fortunately, as of now this is a thing of the past.

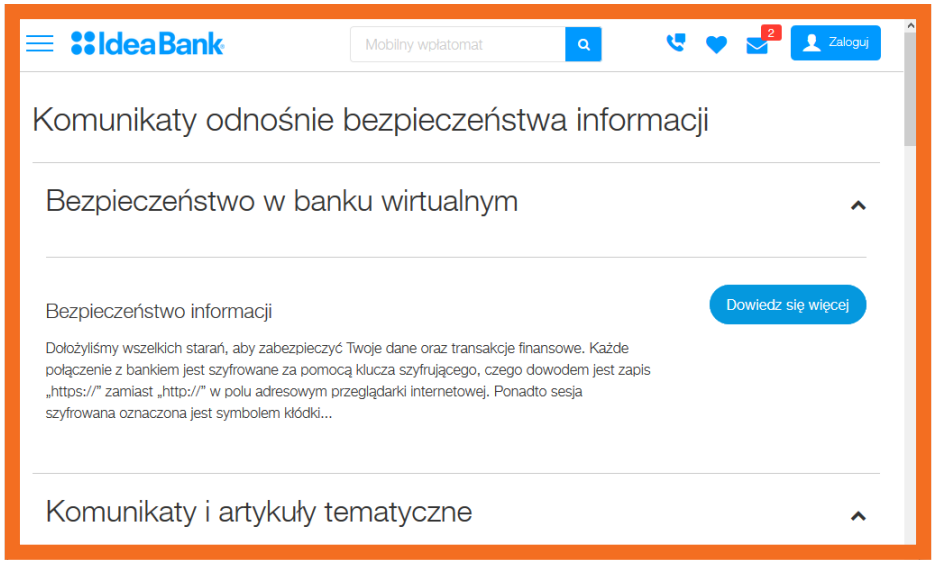
The bph.pl website does not have a secure connection option, because it uses the same address with *https* to open the log-in page for the transaction system. Whereas, an attempt to go to https://raiffeisenpolbank.com will never be successful, because the connection is impossible.

What about banks which cannot make their mind up about choosing a single domain? The **Orange Finanse** brand uses the domains: orangefinanse.com, orangefinanse.pl, www.orange.pl and orangefinanse.com.pl. The first two of them will not load through HTTPS due to the inconsistency between the domain name and the certificate, the third redirects us to HTTP, only the fourth one works correctly (however, it is a log-in page). Neither the citibank.pl nor citihandlowy.pl domains work with HTTPS, similarly to citibankonline.com (yet again – name inconsistency).

Let's be frank: the people responsible for the state of affairs described in this chapter should be very alarmed. These days we expect banks to use a mandatory HTTPS connection and an Extended Validation certificate for all websites listing their services, not to mention transaction pages. At the time of writing this article, exemplary websites belonged to **mBank**, **Millennium**, **Alior Bank**, **Idea Bank**, **Eurobank** and **Credit Agricole**.



An attempt to launch https://www.citibank.pl



The "Learn more" button redirects us to a PDF document loaded up through an unsecured HTTP protocol.

How to find a bank app in the Play Store?

This may seem like a banal question – we have to find the link on the bank's website or use the store's search engine. But is it really so simple? The Play Store is quite infamous for its rather relaxed approach to abuse – it does not really do much to prevent it, since actions are only taken after the damage has been done. An attempt to log into a substituted app may have terrible consequences, so it is best to get it right the first time round. The name of the service can be useful here, as it is unique for the entire store and visible in the Google Play URL address.

Some domain names correspond directly to application suites:

- mbank.pl – pl.mbank,
- eurobank.pl – pl.eurobank.

Sometimes it is close enough, for example:

- bzwbk.pl – pl.bzwbk.bzwbk24,
- pkobp.pl – pl.pkobp.iko.

Unfortunately, sometimes the name of the suite advertises the app developer rather than represents the bank:

- pekao.com.pl – eu.eleader.mobilebanking.pekao,
- bgzbnpparibas.pl – com.comarch.mobile.banking.bnpparibas.

In some extreme cases, the name of the suite says nothing about the bank:

- com.konylabs.cbplpat to CitiHandlowy

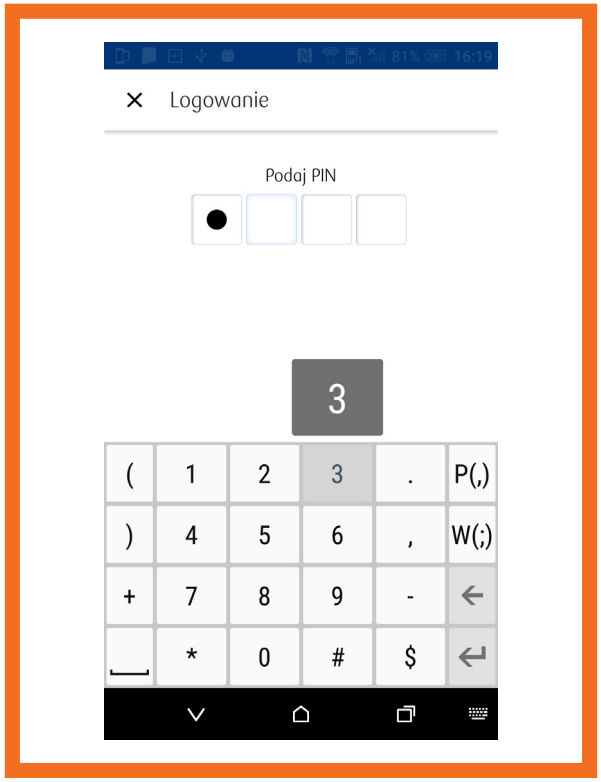
or even proves to be misleading:

- alior.bankingapp.android to... T-Mobile Usługi Bankowe.

It is clear why it does not take much to make a fatal mistake in these circumstances.

Screenshots and more

Android allows the user to save screenshots. In general, this function is either triggered by pressing the right combination of buttons or hidden in the system menu. Standard programmes cannot gain access to contents displayed by other apps. This is probably why the **PKO BP**, **Pekao**, **mBank**, **Orange Finanse**, **Raiffeisen**, **Citi**, **BPH**, **Eurobank**, **ING**, **Alior Bank**, **T-Mobile Usługi Finansowe** and **Bank Smart** apps do not block the screenshot function while logging in, even though shots of the on-screen keyboard clearly show the sequence of the buttons pressed. In the case of **Idea Bank**, the password will not leak, but subsequent screenshots will show the other data visible on the screen.



A series of screenshots will reveal the user's PIN code.

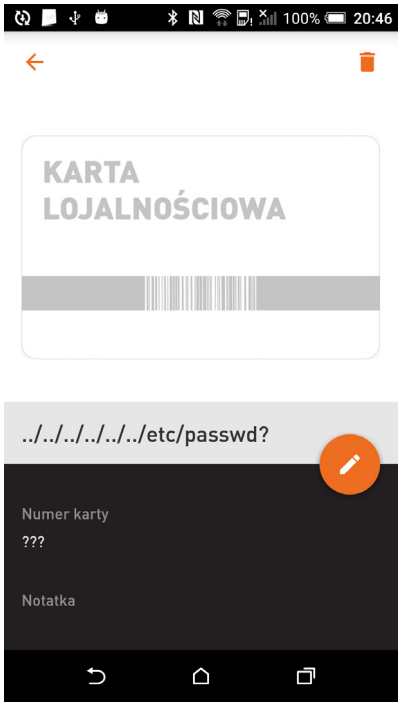
However, the screenshot function is blocked by the **BZ WBK**, **Getin Bank**, **Millennium**, **BGŻ** and **Credit Agricole** apps.

This may seem excessively cautious, if it was not for the precautionary case of Alcatel A564C. The manufacturer included the Qualcomm SystemAgent[5] diagnostic kit in the Alcatel A564C software (probably by mistake). This software allows all programmes to take screenshots without any special permission; provided that the developer of the screenshot app had not applied any active prevention means.

Another issue found in the **PKO BP** app is the deserialisation of untrusted data provided with the app launch command. This enables the attacker to inject an object, which is later used by the application. Such errors may lead to the execution of an unauthorised code.

In the loyalty card module, the **BPH** app used the name provided by the user as part of the file name, while it was possible to enter, for example, a sequence of characters ../../../ (moving between catalogues). Placing blind trust in the user is bad practice. As a result, a completely different file will be read instead of a picture of the card.

In some of the applications, we were concerned about the high number of libraries and components attached – sometimes over twenty items per app. Third party components can be very helpful to programmers; this means however, that the ready app is a sum of errors and vulnerabilities of its own code and all of its components.



The **BPH** app – use of text entered by the user as part of the file name.

4. https://github.com/rednaga/disclosures/blob/master/QCOMSysAgent.md

Application architecture

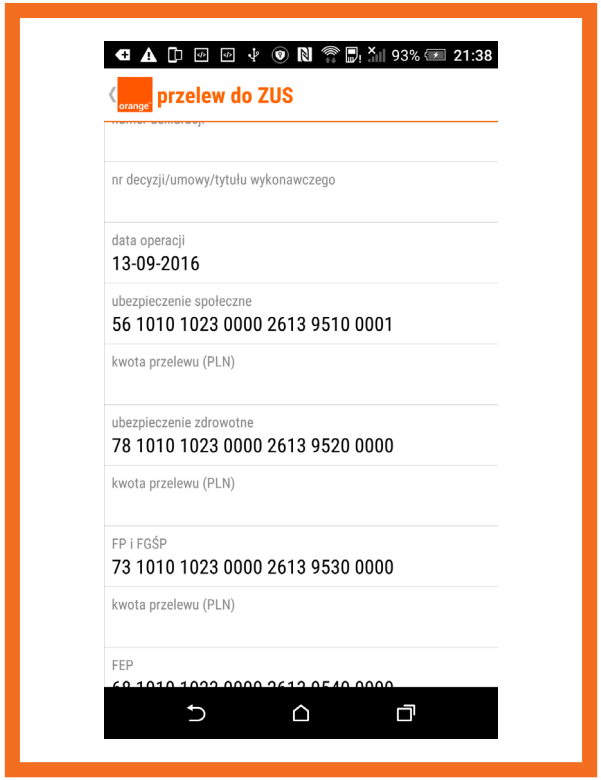
A big problem with the **BGŻ** and **Alior Bank** apps is connecting to the service (programme component operating in the background) by using a name which could be falsely claimed by any other programme. If that programme had been installed earlier, the transfer of data outside the app is unavoidable. A security warning had certainly been sent by Android to the logs, but it was ignored by both developers and testers.

```
<v:Envelope xmlns:v="http://schemas.xmlsoap.org/soap/envelope/" xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xmlns:d="http://www.w3.org/2001/XMLSchema"
xmlns:c="http://schemas.xmlsoap.org/soap/encoding/" xmlns="http://mbank.pl/">
  <v:Header>
    <RequestInformation>
      <AppType>AND</AppType>
      <AppVersion>2.7.3</AppVersion>
      <BankId>1</BankId>
      <SessionId>scSunD-9lEy[redacted]6Gdg==</SessionId>
    </RequestInformation>
  </v:Header>
  <v:Body>
    <ZUSRegisterTransfer>
      <ammount1>11.0</ammount1>
      <ammount2>11.0</ammount2>
      <ammount3>11.0</ammount3>
      <ammount4>11.0</ammount4>
      <creditAccount1Number>56101010230000261395100001</creditAccount1Number>
      <creditAccount2Number>78101010230000261395200000</creditAccount2Number>
      <creditAccount3Number>73101010230000261395300000</creditAccount3Number>
      <creditAccount4Number>68101010230000261395400000</creditAccount4Number>
      <decisionNumber></decisionNumber>
      <declaration>072016</declaration>
      <declarationNumber>01</declarationNumber>
      <documentNumber>A[redacted]01</documentNumber>
      <documentType>1</documentType>
      <email1></email1>
      <email2></email2>
      <isItFirstCall>true</isItFirstCall>
      <nipNumber>[redacted]</nipNumber>
      <payType>S</payType>
```

Contents of an intercepted (and stopped) message with a ZUS Social Security transfer order.

While examining the work files we noticed that some applications were parametrised with data from databases or XML files. Unfortunately, the integrity of such collections is not protected. We found vulnerabilities in the **Millennium**, **mBank** and **Orange Finanse** apps. We modified files (a simulated effect of a successful attack), which resulted in a bogus social insurance account number being displayed in the Social Insurance Institution (ZUS) form. The issue did not only concern the presentation of data – **mBank** and **Orange Finanse** apps actually tried to make the transfer! We sadly had to stop the transmission of data right then and there not to expose the servers to the risk of unauthorised tests.

Some apps communicate with various IT systems within the same bank, with different addresses, protocols, and levels of security. This is not recommended. It means that the entire mobile service is only as secure as its weakest component.



Swapped ZUS social insurance number in the **Orange Finanse** app.

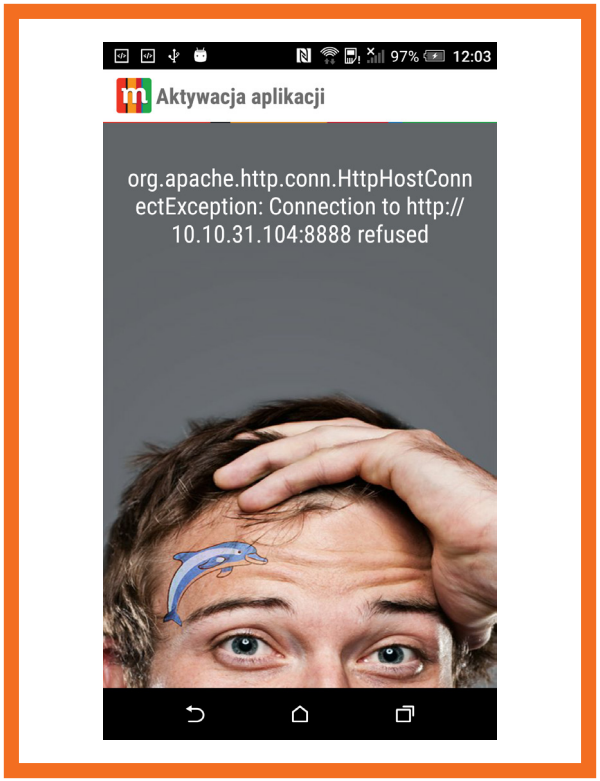
Poor app quality control

While examining the applications, we encountered errors which should have been detected at an early stage of internal testing. We need to stress that, while collecting the material for the report, we only spent a few hours on each app, which is much less than regression tests of each new version should undergo, under normal test circumstances.

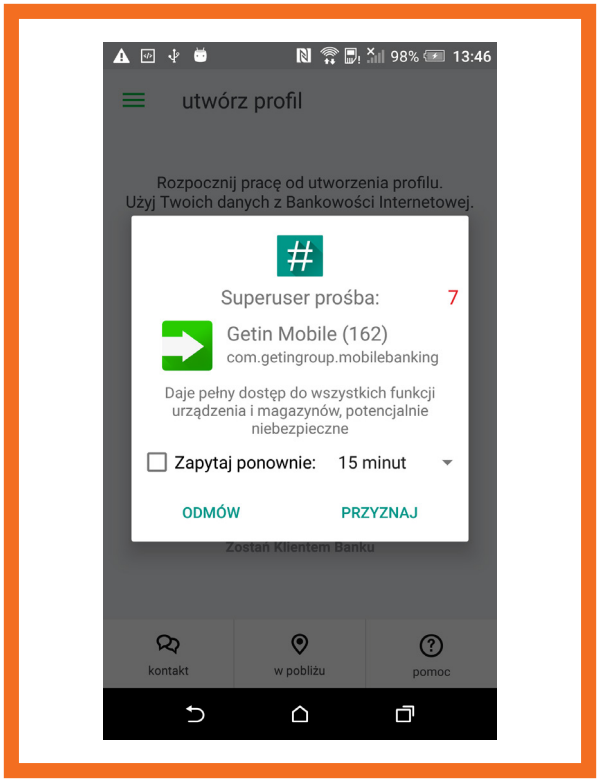
So:

- after leaving the map view, the **BGŻ BNP** app did not revoke the GPS location subscription, which caused the battery to drain in just a few hours
- the **BPH** app used a library to read the number from a photo of the card which was outdated by over a year, which meant that the function did not work at all on newer phones
- the older version of the **BPH** app was unable to inform the user about a forced update, instead, a server connection error message was displayed
- a forced update of the **Idea Bank** app could be skipped by going to the Play Store and then back by pressing the *back* button
- the **Smart Bank** app had difficulties with the layout of various components on the screen, which meant that 150 entries per second were being made in the logs and the phone was overheating
- in case of connection issues, the **mBank** app displayed the exception contents in English, which were impossible to understand for an average user
- the **Getin Bank** app made it to the Store with inoperative maps
- although the **Millennium** and **BGŻ** apps blocked network connection attempts made with a fraudulent certificate, they had issues with restarting work after switching to another, safe Wi-Fi network

The **Pekao** and **Raiffeisen** apps (both made by the same developer) contained diagnostic messages with multiple typos, which would have been found at the code review stage in a normal production process. Exception descriptions, such as, „You do something wrong mapper must contains item type for item %sat this place, Maybe You modify your adapter in illegal way !?" may indicate lowered standards of work on the source code.



Incomprehensible error message in the **mBank** app.



On rooted devices, the **Getin Bank** app needlessly tries to increase the privilege level.

Debug code

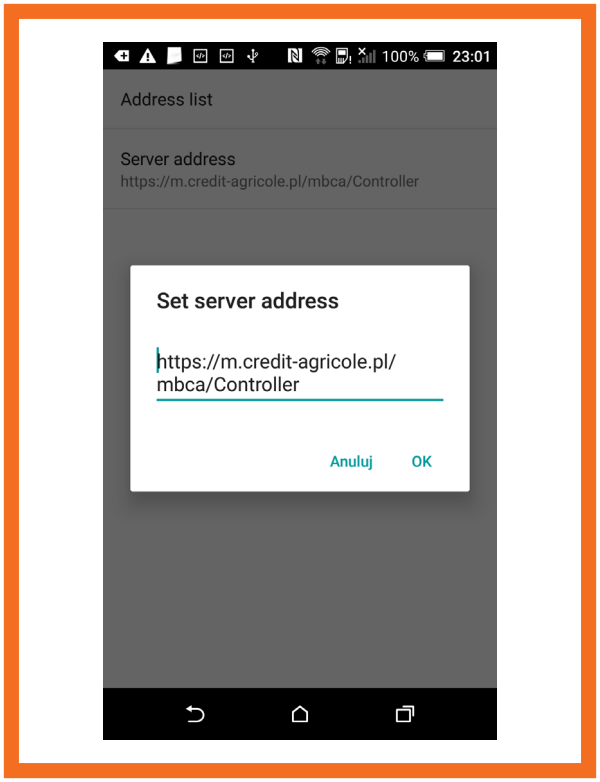
On a technical note

Debugging – means removing software errors. A debug code comprises programme parts used for diagnostic purposes, which are no use to the end user. It is a good practice to ensure that the debug code is not included in the final version of the app released publicly.

While working on an app, programmers add various auxiliary structures to the source code to make it easier to develop or test the application. An example of the above is the mechanism of selecting the server which the app should communicate with. While writing code, developers usually use test servers, and later integration and acceptance testing environments are used for that purpose. Production servers are only used at the very end – so the switch function may be useful while diagnosing application errors.

Sending an app to the store which allows server address changes may have catastrophic results. An attacker can force communication with their device, intervene with the communication between the device and the server, and at the critical moment, for instance, change the account number and transfer amount. As we found, that was possible in the **Credit Agricole** app, which blindly accepted any new address provided in the substituted XML file.

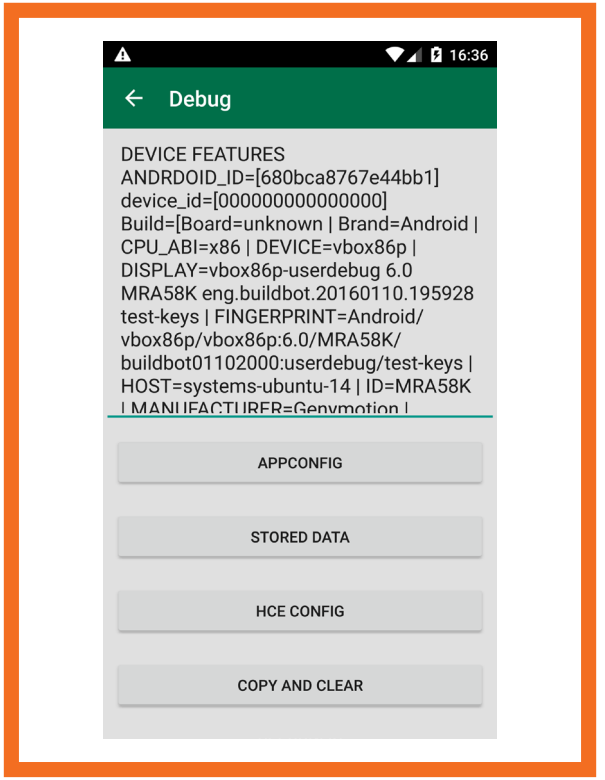
Theoretically speaking, in order to be able to create or modify the settings of another app, an attacker needs to gain superuser privileges in the hijacked device first. However, all observant readers will remember the vulnerabilities of Samsung keyboards and Alcatel's epic failure with the Qualcomm SystemAgent software described earlier.



Debug code in action – server address change in the **Credit Agricole** app.

The latter meant that all programmes were able to overwrite other programme files, which means that an attack on **Credit Agricole** would be ridiculously easy – like taking a candy from a baby.

The example above shows why all auxiliary mechanisms must be removed from the final production versions. Test environment addresses (found in the **Pekao**, **BZ WBK**, **mBank**, **Raiffeisen** apps) can help cybercriminals identify weaknesses in the infrastructure. Automatic form completion data reveal how many characters the test passwords have (**Credit Agricole** again). We also managed to activate the Leak Canary diagnostic library, which is part of the Eurobank app. The library developers clearly state that it should never be included in the final version, and describe a simple method which would prevent that from happening. The **BZ WBK** app contained, among others, contactless payment diagnostic tools.



Debug functions of the **BZ WBK** app.

Even if a debug code does not pose a direct threat, it may provide an attacker with information required to make the attack in another way.

Application package contents

The APK file with the Android app is simply a ZIP archive containing files distributed according to detailed specifications. We examined the contents of all applications, as we suspected that the case of a password file described earlier was not an isolated incident and that we would be able to find other interesting information there. Our intuition was not wrong.

The **Bank Smart** app contained a develop.properties.example file with a detailed description of all test operations, which could be activated while compiling the programme. Things get awkward when we look at the list of test accounts and see names such as Hans Kloss and James Blond side by side with... Putin.

The **BPH** app carried around a dozen unused KML files (which the users of, for example, Google Earth, would be quite familiar with). In the **Credit Agricole** app we could find Jenkins logs (a continuous integration tool), which contain some clues about the infrastructure used to develop the APK suite. In the **ING** app we found an incorrectly integrated Facebook library. One of the components of the **PKO BP** app clearly indicates the last name of the developer who had worked on the project (and at the same time confirms that the project had been outsourced). In the **Citi** app, we found an image template of... a Chinese transfer confirmation.

It is clear to see how often APK files are thrown into the Play Store without undergoing even a brief inspection of their contents. We add this point to our collective complaint about insufficient app testing.

Application reception by users

In the Play Store, users give the highest scores to the **PKO BP**, **BZ WBK**, **mBank**, **ING** (older version), **Millennium**, **Orange Finanse** apps – they all received a score of at least 4.5 stars out of 5.

The **Moje ING** app, released in the second half of 2016, proved to be a spectacular failure, receiving only 2.1 stars (with one star being the most popular score). In the comments section, users complain about the inability to use the software on two devices, the inability to access a private and corporate account at the same time, slowness, and other defects, which rise to the surface when using a weaker Internet connection.

II. CONTACTING THE BANKS

All defects and vulnerabilities described in the first part of the report were found from the position of a customer who has opened an account, installed the mobile app, and began to check what standard of service they are dealing with. The banks were not informed about the start of testing, so it was not possible to carry out standard penetration tests.

To stay on the light side of the force, we did not check the vulnerability of servers and did not modify outgoing traffic from the app to the server. The same goes for the apps – we checked their resistance to potential attack vectors (for example, data files modification) and examined their structure, but did not modify the executable code or affect the memory of active processes.

In terms of publishing the test results, we followed the unwritten rule of *responsible disclosure*, which means that we informed the banks about the results 4 to 10 weeks (with one exception of 3 weeks) in advance.

Disclosing vulnerabilities in practice

On a technical note

PGP (and the free GPG version) are tools which use so-called public-key cryptography, which is – in very simplistic terms – a mathematical method of disguising messages in such a way that messages encrypted with the first of two keys can only be decrypted with the second key. The first key can be disclosed publicly (it is a file with a long character sequence) and the second must remain secret.

Therefore, if a bank discloses the public key of its security department, we can use it to encrypt messages sent to the general customer services address and be sure that only employees of that department will be able to read its original contents.

Although we only encountered critical vulnerabilities in a few cases, we decided to treat all reported information as confidential. We started by calling all customer service centres and asking to be put through, or to be called back by the security department. We treated customer service centres (contacted by telephone, via e-mail, or contact forms) as untrusted parties. We are aware that this may be an unfair approach, but taking into account how common outsourcing is in this area, combined with high staff rotation, low pay, and overall cost cutting practices – we decided not to share information about the vulnerabilities of IT banking systems with them.

The telephone contact method failed completely. Not a single contact attempt was successful. Customer service centres are completely separated from business operations, and are unable to escalate the matter to persons responsible for IT security. The staff tried to be as helpful as they could, but discussing information about vulnerabilities as if making a complaint is not a path to success.

In our second attempt, we decided to search bank websites for information about the procedures of reporting faults and vulnerabilities. Another fail! None of the websites mention them. Later it turned out that the international website www.ing.com features a PGP key, a procedure to be followed by testers and information about the rewards offered for the responsible provision of information.

In our third attempt, we used the conventional e-mail (or an online form, if there was no e-mail address; while the **BGŻ** form did not allow us to submit the message without consenting to receive spam). The message reads as follows:

I would like to get in touch with your bank's security department. As I was unable to find the right information on your website, I would like to ask:

- *Do you have a public contact procedure regarding vulnerabilities of your website and mobile application to external attacks?*
- *Do you have a declared response time in which your specialists respond to such contact attempts?*
- *Do you have a public PGP/GPG key which can be used to confidentially exchange information with the security department?*
- *Do you run a "bug bounty" programme which offers rewards to individuals who report vulnerabilities responsibly?*

I will be grateful for your reply and for an indication of the way in which information can be exchanged confidentially with the security department.

The above message was sent out to the banks on Thursday, 30th June, at about 3p.m. The time of day is specified, because it should be noted that we received a response (with PGP keys) on the same day from **Alior Bank** and **Milennium**. **ING** responded the next day, directing us to an instruction published on the corporation's international website. On 4th July, so after the weekend, we got a reply from **mBank**, its clone **Orange Finanse**, and **BZ WBK**. A week after having sent the enquiry, we received the public key for **BPH**. Overall, we received a response from less than half the banks.

Later on, things got more difficult. And a bit stranger.

Please await our response in this matter

We purposely withheld the bank names below in order to protect innocent employees who may have otherwise been reprimanded, for what is clearly the result of a lack of appropriate procedures.

In one case, the response described a primitive method of ensuring confidentiality. The bank instructed us to send an archived ZIP file to the customer service centre's address, and then to send the password via text message.

One of the banks remained silent.

Another bank kept promising us that we would receive a response from a specialist responsible for issues needing further clarification. And then kept silent.

And yet another bank was silent for a while, and then asked for a ***sample in order to verify our report***.

Desperate times call for desperate measures. Hence, as our last resort, we used the LinkedIn website (an e-mail to a board member works miracles) and reached out to PR representatives (in this case, miracles took more time and were less spectacular, but in the end, they still happened).

Consequently, all of the banks received the message in one of several ways: it was sent to the customer service centre's address after having been encrypted with the PGP key, through an online form in the corporate mailing system, or – in case of no critical vulnerabilities – it was sent to an individual e-mail address of the bank officer handling the case.

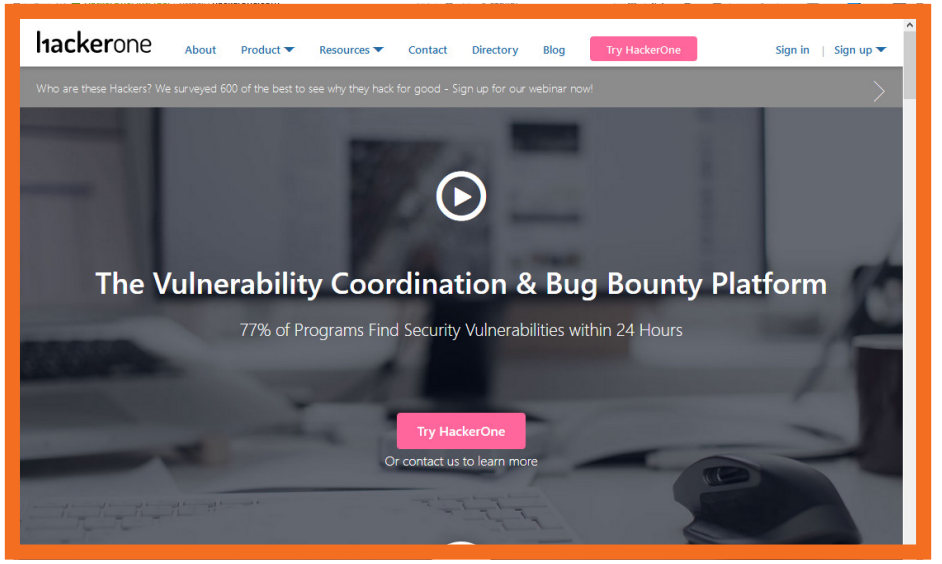
Dear banks! You need to improve the process!

It is in your best interest to simplify the procedure of reporting security-related issues. A lack of a fast-track process may result in painful consequences in the future. You need procedures which ensure that a response can be provided within 15 minutes, irrespective of the day of the week. Publish guidelines on your website, publish emergency phone numbers and e-mail addresses, define a ***critical vulnerability pathway*** for customer service centres.

Bug bounty

In many companies from the IT sector it is a custom to thank individuals who report vulnerabilities – for example, by publishing a thank-you note on the website, giving out material prizes, and cash rewards (in the case of Microsoft and Facebook – even over $10 000).

Among the eighteen banks we informed about the results of our application tests, only **Bank Milennium** issued a written thank you letter with a gift in the form of a Yubico authentication key.



Hackerone – a website which supports co-operation between companies and institutions with independent researchers.

III. POLISH BANKS, AMERICAN SERVICES

Crashlytics

During the tests we noticed that the mBank, Orange Finanse, IdeaBank and Bank Smart apps use the Crashlytics service, owned by the American company known as Twitter.

Crashlytics is an automatic issue reporting tool. If the app *crashes*, the details of the problem are sent to Twitter servers. The programmer can use a convenient panel to view not only the stack of function calls which caused the crash, but also statistics – how many times a specific type of error has occurred, how many users have encountered it, at what memory usage, etc. It is a great tool, but we are worried about one important detail: it connects a POLISH financial institution to an AMERICAN server.

How much information can be found in an error report? It may not seem like much, but the devil is in the metadata. If the app reports an error in the MortgageLastStepSuccess.java class (the name suggests that a mortgage has been granted), Twitter only seemingly does not find out anything about the phone user. In practice, however, it can take note to send adverts related to interior design and household appliances to that IP address.

External libraries may also turn out to be a surprise – for example, two years ago Google programmers extended the range of diagnostic information in the GSON library. Previously, an error message entered in the report would read, more or less, as follows: *error while processing »100000CHF« text into a number*, after the changes, the same *error will produce an entry that reads: error while processing »100000CHF« text into a number in node »User› Loans › Mortgage[2] › Balance«*. Right there, that is another leak of the user's financial data. This example shows that even if everything is okay today, a routine upgrade of external components may suddenly change that, (and let's be honest, nobody does library change audits).

If the user is asked for permission to send data after a failure (**mBank** asks, **Bank Smart** and **IdeaBank** do not), the contents of the question still do not indicate that the information is sent to a server in the USA, and not directly to the bank.

We asked Dr Paweł Litwiński, an attorney specialising in personal data protection, new technologies, and telecommunications law, to comment on the above issue:

We should ask ourselves whether such a mode of operation of mobile bank applications using Crashlytics/Fabric.io, is in accordance with the law. Even more so since the issue of legal compliance in case of (personal) data transfers to the USA has been widely discussed over the recent months (see: judicial decision, dated 6 October 2015, regarding Maximilian Schrems, C-362/14).

In principle, the transfer of personal data to the USA is prohibited. In order for it to be legal, entities involved in the transfer need to use certain legal instruments designed specifically for such situations, i.e.:

- *to conclude a mutual agreement based on model clauses, i.e. a template agreement approved by the European Commission, or*
- *to operate within one capital group on the basis of corporate rules.*

Now, American data recipients can also join the Privacy Shield programme, whose members can receive data from the EU.

Twitter, Inc. has not joined the Privacy Shield (which can be easily verified online). It is not part of the same capital group as ***mBank*** *either – so the only solution is an agreement based on model clauses. Has such an agreement been concluded? We do not know and have no real way of finding out – but it needs to be stressed that this is the only available method of legalising the transfer of personal data to the USA, in the case examined here.*

But, we should take note of one more thing – we do not really know if any personal data is transferred to the USA. This, in turn, poses a question as to what the above mentioned banks know in this regard? This is a key question, as personal data administrators (i.e. a bank with regard for its clients) should exercise due diligence to ensure protection of personal data (art. 26 item 1 personal data protection act of 29 August 1997). Due diligence requires that each case, in which another entity may gain access to personal data, is identified and then allocated the right legal basis, or is eliminated. It is especially important that information contained in a mobile app regarding the user's finances is protected by bank secrecy – which imposes even more restrictive rules on its disclosure than the personal data protection act.

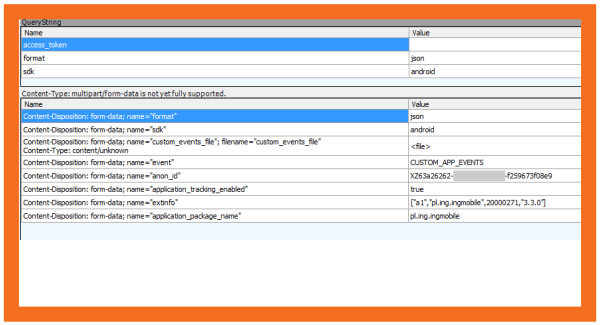
So what – from an attorney's point of view – should an entity which intends to use a mobile app, which bears the risk of transferring its users' personal data to other entities, do? Firstly, all potential cases of such data transfers should be identified. Secondly, if no legal basis has been identified for the disclosure of personal data, such cases should be eliminated – this is a requirement imposed by the personal data protection act.

Facebook and others

Does a Polish bank have the right to inform the American Facebook Inc. corporation that its client has just started using a mobile application? Without informing the client, without allowing them to disable this function, and without including any mention of the *Facebook* brand in its regulations? That is exactly how the **ING** app operates (the older of two versions) – it obediently serves the popular social network by providing data about who, where, and when starts a session with the app.

In other applications, we found codes from Gemius (**mBank**, **Orange Finanse**), Adobe Marketing Cloud (**ING**, **Citi**), Webtrends (Millennium), SponsorPay/Fyber (Bank Smart), HockeyApp (BPH). To be absolutely clear – we would not oppose embedding a reporting module in the app, provided that the data was sent directly to the bank.

Maybe one of our readers could provide us with a more comprehensive legal analysis of this issue?


QueryString

Name	Value
access_token	
format	json
sdk	android

Content-Type: multipart/form-data is not yet fully supported.

Name	Value
Content-Disposition: form-data; name="format"	json
Content-Disposition: form-data; name="sdk"	android
Content-Disposition: form-data; name="custom_events_file"; filename="custom_events_file" Content-Type: content/unknown	<file>
Content-Disposition: form-data; name="event"	CUSTOM_APP_EVENTS
Content-Disposition: form-data; name="anon_id"	XZ63a26252-[redacted]-f259673f08e9
Content-Disposition: form-data; name="application_tracking_enabled"	true
Content-Disposition: form-data; name="extInfo"	["a1","pl.ing.ingmobile",20000271,"3.3.0"]
Content-Disposition: form-data; name="application_package_name"	pl.ing.ingmobile

Data sent to Facebook at the launch of the **ING** app.

SUMMARY

As shown in part I of this report, the security and quality of mobile apps released by Polish banks do not meet today's standards. More can and should be expected from financial institutions. As a result of the growing popularity of mobile banking, the number of ways in which it is possible to attack infrastructure, user data, and financial resources is increasing.

In part II, we described the obstacles we encountered while reporting the detected vulnerabilities to the banks. It would not take much to change things for the better in this regard. In fact, all that is needed is an additional scenario to be used by the customer service centre, a PGP key published on the website, a designated e-mail address for reporting problems.

The topics discussed in part III are only a brief introduction to the complex issue of the circulation of ***sensitive data*** between the big players on today's IT market. Banks should make every effort to avoid feeding big data systems with information about their customers.

This report does not exhaust the issue of the security of Polish mobile banking applications. We completely omitted the iOS system. We ignored the area of NFC/HCE contactless payments. We passed over the risk associated with the use of text messages as an authentication method. It is necessary to continue the research and to keep an eye on banks in an attempt to ensure that the mobile products on offer do not raise as many objections as they do now.

TESTED APPLICATIONS

- PKO Bank Polski (pl.pkobp.iko)
- Pekao SA (eu.eleader.mobilebanking.pekao)
- Bank Zachodni WBK (pl.bzwbk.bzwbk24)
- mBank (pl.mbank)
- ING Bank Śląski (pl.ing.ingmobile oraz pl.ing.mojeing)
- Getin Noble Bank (com.getingroup.mobilebanking)
- Bank Millennium (wit.android.bcpBankingApp.millenniumPL)
- Raiffeisen Polbank (eu.eleader.mobilebanking.raiffeisen)
- Citi Handlowy (com.konylabs.cbplpat)
- BGŻ BNP Paribas (com.comarch.mobile.banking.bnpparibas)
- BPH (pl.bph)
- Alior Bank (com.comarch.mobile)
- IdeaBank (pl.ideabank.mobilebanking)
- Eurobank (pl.eurobank)
- Credit Agricole (com.finanteq.finance.ca)
- T-Mobile Usługi Bankowe (alior.bankingapp.android)
- Orange Finanse (com.orangefinanse)
- Bank SMART (pl.fmbank.smart)

NO MOBILE APPLICATIONS

- Deutsche Bank Polska
- Santander Consumer Bank
- Volkswagen Bank
- Bank Pocztowy

NO MOBILE TRANSACTIONAL APPS

- SGB Bank
- BOŚ Bank

PGS SOFTWARE

About the report

Concept and preparation: **Tomasz Zieliński,** tzielinski@pgs-soft.com
Editor: **Krzysztof Piskorski,** kpiskorski@pgs-soft.com
Typesetting: **Michał Cichoń,** micichon@pgs-soft.com