Secure Internet Banking on Insecure Hosts

Christos Tziortzios - christos.tziortzios@os3.nl
University of Amsterdam
System and Network Engineering (MSc)
(Dated: July 31, 2012)

This paper discusses using one - time applications for secure and usable Internet Banking. The authentication and authorization methods are those implemented by ABN - Amro bank in the Netherlands. Customers use a secure device and their smartcard to generate the one - time codes needed to login and sign their transactions. This paper suggests that one - time Java applets can be used for Internet Banking transactions. Although there are more secure schemes, banks need to balance security and usability; secure solutions that customers are not willing to use can be disastrous from a marketing perspective. The goal of the scheme is to stop the Man - in - the - Browser attack as it is implemented at the moment of writing. Moreover, new attacks should be expected to arise against it; these attacks should be hard to automate. An important trait of the scheme is that it is suitable even for environments, over which the user has limited or no control, such as internet - cafe computers. Moreover, banks can easily update the application in case of a security breach.

I. INTRODUCTION

Internet Banking (IB) services have enabled bank customers to perform their transactions from the comfort of their homes or while on the move providing a 24 hour a day, seven days a week service. These services no longer require physically going to the bank. This saves both customers and banks time and money. Furthermore, IB and the ways in which it is performed can be used as marketing tools, giving certain banks a competitive advantage. This has become more obvious with the introduction of Mobile Banking applications for Smart phones.

Ever since the first IB services have been offered, banks have been confronted with the obvious exposure to Internet criminality. Ever since the introduction much attention has been paid to ensure security of online transactions. There is still an ongoing "cat and mouse" game between banks and criminals [1, 2]; banks implement new security features and criminals continuously try to find the weakest link and compromise that. As a result, there is an evolution in the security features of IB applications, which leads to more advanced attacks. Increased security, however, often means limiting usability [3]. As a result banks are confronted with a constant struggle for the optimum balance between usability and security.

IB attacks have evolved from simple keyloggers, which would capture users' passwords to Man in the Middle (MitM) attacks. Nowadays most European banks use two-factor client-side authentication for IB [4]. This is similar to what most people are used to when using an Automated Teller Machine (ATM); authentication is based on something you have (card, cell phone) and something you know (pin code).

Although the servers on the bank side can be thought of to be trusted, security of hosts used on the customer end as well as the network (Internet) cannot be trusted. Banks take for granted that a lot of their customers perform their transactions using computers, which are infected with malicious software. Currently the most severe type of attack is the Man-in-the-Browser (MitB) attack. A Trojan Horse is installed in the client side, which intercepts and manipulates calls between the browser and its security mechanisms [5]. What should be noticed is that the target browser will not provide any warnings related to Public Key Infrastructure certificates; there is no Man-in-the-Middle in the connection between the client host and the bank server. The outcome of the MitB attack is that the bank customer will see the correct transaction in the browser, while the bank will get a transaction that transfers money to an account of the criminal's preference. While it would be extremely hard for a bank customer to detect the fraud, the bank side can sometimes detect possibly - fraudulent behaviour. However, the high sophistication and the constant evolution of the attacking tool let the attack succeed in many cases.

A. Related Research

A lot of research has been performed, related to secure IB. Weigold et al. [6] state that malware on a compromised host is capable of faking all information displayed on the computer screen, including digital certificate manipulation. They also indicate that using a trusted device is necessary for secure e - commerce. Weir et al. [7] illustrate the usability - security tradeoff. In their survey about two thirds of the sample would rather use an authentication scheme that they perceive as possibly less secure, as long as it is more usable. The fact that banks in the Netherlands tend to compensate victims of fraud in most cases, in fear of reputational damage, makes IB users reluctant to accept less usable schemes. Hanacek et al. [1] summarize attacks against IB, focusing on the authentication and authorization problems. Oppliger et al. [2] distinguish three types of client - side attacks: credential - stealing, channel - breaking and content - manipulation attacks; the MitB attack is a content ma-

nipulation attack. Among other things, they suggest that in order to provide transaction authentication, the user can input the transaction details in the web application. In addition to that, the user will enter the same data in the trusted device, which will generate Transaction Authentication Numbers (TAN). Then the user will enter those numbers in the web application. This requires that the user inputs the transaction data twice. Moreover, the user must input the TANs. This makes this scheme less usable, but much more secure. Provided that the bank customer got the destination account from a trusted source, it is less likely that he will be tricked into entering the wrong transaction data to the secure device. As a result criminals will not have the TAN for the fraudulent transaction.

Several different concepts have been examined in order to mitigate the risks related to IB. Especially in the case of the Man - in - the - Browser attack different technologies have been examined [8]. Portable, hardened web-browsers and Live - CDs are examples of that. Such solutions may be suitable for home usage, even though they require higher technical knowledge levels at the customer end. However, they may not be functional when the customer uses a computer in an Internet - cafe and hosts on which there is limited or no control by the customer (e.g. USB ports may be disabled). Moreover, the fact that there is no malware on the Live distribution used, does not mean that there are no vulnerabilities in the, otherwise trusted, Operating System [9].

Another option involves sending out - of - band messages; an SMS is sent to the customer's cell phone including the details of the transaction to be performed and an authorization code. This approach requires that the cell phone is free of malware. This is not always a valid assumption, especially nowadays that smartphone users install all kinds of applications on their cell phones and circumvent the security features of their device. It is also important to notice that users do not always pay attention to the transaction details sent through the SMS [10], thus even if the users can see the fraudulent data on the SMS, they could still authorize the transaction.

From a security point of view, using a secure device to authorize transactions seems as the most promising approach. Any software - only solution cannot be trusted when the customer uses a compromised host. Trusted secure devices can provide increased security. However, this approach provides limited usability; the customer must have the secure device any time he wants to perform a transaction. Thus, there must be a balance between security and usability.

B. ABN Amro IB scheme

In order to log into ABN Amro IB, customers have to use one - time passwords (OTP), generated on a trusted device by a smartcard; the card that customers use for their transactions on an Automated Teller Machine (ATM). At the moment of writing, customers can either use e.dentifier or e.dentifier2; both are secure devices, distributed by the bank. Figure 1 provides a simplified overview of the login and transaction procedure. In order to generate the codes, users must enter their PIN code. When customers want to perform transactions, they enter the transaction details to the web application and send them to the bank server. Next the bank server generates a receipt and a challenge. If the transaction details are correct, the user will input the challenge in the secure device and get a response. Finally the user will input that response to the web application in order to digitally sign the transaction.

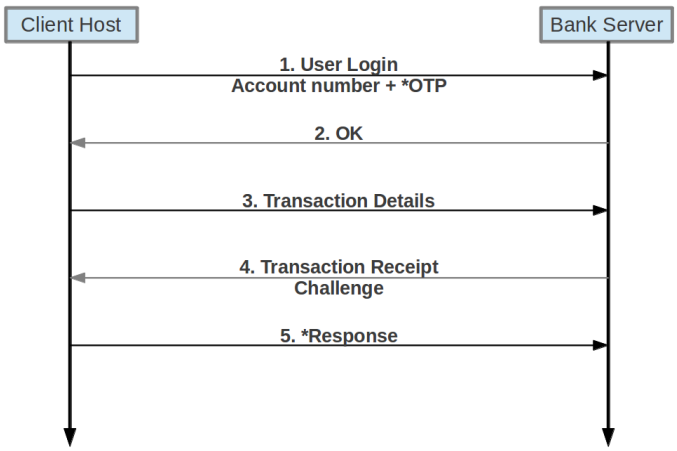

FIG. 1: Simplified Login and Transaction Procedure; Tasks tagged with "*" require e.dentifier

e.dentifier2 can be used either connected to the user's computer through a USB port or not connected. When the user connects this device to his computer, the integrity of the transaction details displayed on the screen can be considered guaranteed; this is the "See What You Sign" principle. Customers are also allowed to use the device in a non - connected mode. This is the easy thing to do, since in order to use the connected mode, customers need to install software. Customers tend not to use the connected mode to perform their transactions, either because they are unaware of its increased security or because they find it less usable. Moreover, using the connected mode is not always possible. In certain cases customers have no control over the device they use for their transactions. One example for that is using a computer in an Internet cafe. In this case, users usually cannot install software, or even use USB ports.

C. Research Question

The USB - connected e.dentifier2 scheme is as secure as possible, but not always usable. The scheme that is proposed targets increased security without need to connect a secure device to the computer in use or need for administrative privileges in order to install software on the user's host.

This project introduces an alternative approach for secure Internet Banking transactions: every time an Internet - Banking user needs to perform a transaction, a "one-time" application will be created by the bank server and be run by the client.

The considerations mentioned above lead us to the following research question:

> *Is the use of one - time applications for Internet Banking a secure and usable solution?*

Related to the formulated research question the following subquestions need to be answered:

- What kind of functionality should exist in such an application?
- Which are the risks, related to implementing and using the previously mentioned scheme?
- Which are the strengths and weaknesses of the scheme from a security and usability perspective?

D. Scope

In this project, the concept of using one-time applications for IB is investigated. Due to time restrictions there is not a prototype solution implementing the idea. Moreover, usability will be estimated by the author, based on the complexity of the operations that need to be performed by the user. Last, scalability of the scheme is not investigated thoroughly.

E. Assumptions

The authentication scheme that is currently implemented by ABN-Amro Bank is considered secure. Furthermore, the server-side operations are considered secure.

II. THEORY

In this section the required theoretical background for understanding this report is provided. We give theoretical definitions to the most significant terms and concepts used within the report.

A. One - Time Application

This report discusses using an application in order to perform IB transactions. In this application, local code can be used in order to stop a number of attacks as implemented at the time of writing. The concept is similar to that of a hardened browser, but while in the case of a hardened browser one needs to install software or use a portable application on a USB stick, we can use a Java applet that does not need administrative privileges to run. Part of the paper is specific to Java applets, while some concepts can be used on other runtime environments too.

In this report, a one - time application is an application that is built for one transaction. The functionality of the application is the same for all transactions, but features of the application are randomized. The target of the one - time application is to increase the level of security without deteriorating the customer's experience. This can be achieved by making it harder for the attacking software to know which calls must be intercepted in order to perform the attack. Code obfuscation is needed but not enough. The Internet criminals will probably be able to find out which methods to intercept to achieve their goal, this is why randomization is needed. In order to ensure that the application is only going to be usable once, we need to make sure that the application times out on the server side after it is first used or after a certain period of time.

B. Malware

Malware stands for malicious software. There are different types of malware, some of which are related to attacks against IB. Attackers can combine different malware concepts to increase their power over the compromised host. A common type of malware is Trojans; programs that appear to be useful to the victim, while they hide malicious code. Internet criminals can use rootkits in order to hide the existence of malware on the victim host. They can also use backdoors, which allow them to connect to the compromised host and possibly update and reconfigure the malware that is already there. In earlier times, a keylogger would be sufficient to get an IB user's credentials and credit card details. That would allow the attacker to perform fraudulent transactions. Apart from using keyloggers, attackers are also capable of taking screenshots on mouse events. This enables Internet criminals to get their victims' passwords even when graphical keyboards are used. This led the banks to use more sophisticated schemes, using one - time codes. Even if Internet criminals could capture the one - time codes, the codes would be of no use to them in a later time. This also led to more sophisticated kinds of attacks, such as the Man - in - the - Browser attack. The level of sophistication of

malware drives us to the conclusion that there is nothing that can be trusted in the software level.

1. *Rootkit*

Originally, the term Rootkit referred to tools that used to gain administrative access in UNIX operating systems [11]. There are two types of rootkits, user mode and kernel mode rootkits. Kernel mode rootkits have unlimited privileges, which makes them capable of making the infected host practically completely compromised. Rootkits have the ability to hide the existence of malware to antivirus programs. They also can manipulate memory entries.

C. Man - in - the - Middle attack

"An active attack which involves getting on the path between two legitimate users, relaying their messages to each other, and thereby spoofing each of them into thinking they are talking directly to the other" [12]. The MitM attack can also take place in sessions encrypted through SSL/TLS. In such a case there are two SSL/TLS connections established, one between user A and MitM and one between MitM and user B. Banks use PKI certificates to ensure that if there is a MitM attack, the browser of the customer will raise an alert. However, since we can assume that criminals may have full control over a host, we can also assume that it is possible to make a fraudulent website or piece of code look legitimate to the user.

D. Man - in - the - Browser attack

Man - in - the - Browser attack is a content manipulation attack. It is performed by malware that is installed on the computer of the victim. The malware is capable of modifying the transaction data that are sent to the server, without any visible effect that would let victim - user notice being under attack. Currently the attack is automated. This means that as soon as a host is compromised, the criminal does not need to be sitting behind his computer for the attack to succeed.

The malware (for instance a browser extension) will check all websites that the user visits; when it is a website of the criminal's interest the attack will begin. As seen in Figure 2 the bank customer wants to send an amount of money to a certain account. The malware will capture that data and save it for later use. Then it will change the transaction details to what it is configured to, so that it sends the money to an account of the criminal's interest. It should be mentioned that this is done before data is encrypted through SSL. Moreover, it is not a MitM attack. From a network perspective, the customer is communicating with the bank as normal. In the next step of the IB transaction, the bank will generate a receipt of the transaction. This receipt includes the destination account and the amount of money to be sent. In the transaction receipt there is also a challenge, a number to be used as input to the secure device of the customer. The output of the device is the response. Through this response the bank customer signs the transaction. Normally the customer would not sign the transaction if he notices manipulation of his data. However the malware can scan the HTML document for the receipt fields and simply change the transaction details back to what the user entered. The malware uses regular expressions to find which data need to be altered. This way the legitimate user will never notice signing the fraudulent transaction.

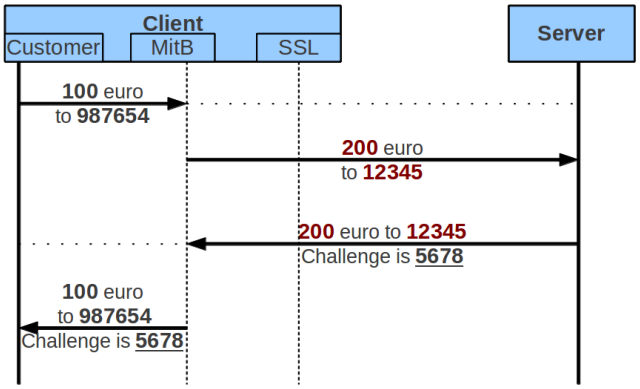

FIG. 2: Man - in - the - Browser attack

Currently, the points of attack for the MitB attack are the following [5]:

- Browser Helper Objects (Internet Explorer) - Extensions (Firefox - Chrome): dynamically - loaded libraries, loaded upon startup. User may be unaware of the existence of the extensions.
- API-Hooking: Malware intercepts calls between the executable application and its libraries.
- Javascript Injection

However, it should be mentioned that if these points are better secured, attack could work on other points as well, for instance by manipulating the memory entries for the data to be sent to the bank server.

III. IDENTIFYING THREATS

According to the proposed scheme, there will be an application, through which IB transactions will be performed.

While there are certain types of attacks against the proposed scheme that are still possible, these attacks should be more expensive that the attacks implemented at the moment.

There are different types of attacks related to IB. About credential stealing attacks, the user authentication scheme will remain as currently implemented by ABN Amro. The application will not implement extra security measures against such attacks. The second type of attacks is channel - breaking attacks, namely MitM attacks. Defending against such attacks requires user's cooperation. Unless users pay attention on certificates, any measures taken by the bank will fail. Adversaries only need to find the weakest link and break it in order to perform their fraudulent transactions; users can be the weakest link. Even applications that automatically check the digital fingerprint of the server will not necessarily increase security, unless users download trusted applications from trusted websites. A low level of protection against MitM attacks can be implemented by making the attack harder to automate, however if the adversary uses his computer in real time, the attack will succeed.

The application may increase security against content manipulation attacks. It must stop current content manipulation attacks and be prepared to cope with new types of attacks. A compromised host can be thought to be infected with any of the malware in the Theory section, as well as with combinations of those. Content manipulation can happen in different steps of the transaction procedure. Adversaries need to manipulate data in cleartext, thus before encryption. This is commonly done through API hooking. Adversaries can also create an overlay application and pass it to the bank customer, while they use the legitimate application for their own transaction. The reason why adversaries need an overlay application is because they need to trick the user into signing the transaction.

Last, there is the threat of a replay attack. Since adversaries may be capable of reverse - engineering the code of the application, they may be able to compromise it and make the bank customer use it. The application should be usable only once by a specific customer at a specific period of time. The server side should be able to check the validity of the application by expecting certain messages from it.

IV. ATTACK SCENARIOS

A. Content Manipulation

Adversaries want to manipulate transaction data transparently to the user. Currently they manipulate calls to OS libraries. Manipulation in a lower level is also possible, but probably more expensive for the adversaries. All communication between client and server is encrypted and attackers need to manipulate data in cleartext, thus before encryption and after decryption. Apart from manipulating calls between the application and the OS, adversaries can also break into the application. One can assume that criminals will be able to reverse - engineer the application, so that they find ways to break it. The outcome of such an attack would be similar to the MitB attack; the customer will not be able to notice signing fraudulent transactions.

B. Overlay Application

Internet criminals need to make bank customers sign their transactions. Instead of manipulating the customer's transaction details, while the customer uses the legitimate application, they can also trick the customer into using a fraudulent application. In this scenario, criminals use the application by the bank, while customers use a bogus application. The criminals need this application in order to make the customer believe he is performing a transaction in order to give them the TAN codes. A possible example is the following:

1. User logs in to Bank website
2. Application is built by the server and downloaded by the host
3. Adversary passes fraudulent application to the user while the adversary uses the legitimate application
4. Adversary extracts the challenge of his transaction and passes it to the fraudulent application
5. Victim - user answers the challenge and signs the fraudulent transaction

The attack can be performed through Javascript injection, but also in other ways, since the customer is using a compromised host. There are certain features in the browsers, such as the same - origin policy, that would not allow this attack to succeed. However, these policies do not necessarily apply on a compromised host.

V. APPLICATION SECURITY CONTROLS

A. Security Objectives

Before defining what the application should do, the goal of the project must be defined. Usability is a key factor in IB; banks could have enforced more secure schemes, but this would probably drive their customers to other banks. Thus, balance between security and usability is needed.

Provided that a host can be completely compromised by malware, it is rational to assume that no scheme can be totally secure if a compromised host is the only device used on the client side. Although any scheme will not be totally secure, it can be more secure than the scheme in use. At the moment of writing, MitB attack is automated. A reasonable goal is to make the attack harder to automate, while keeping the level of usability high. The new scheme aims to increase security by stopping fraudulent transactions. There are no countermeasures against other security related concerns, such as confidentiality of transactions; one can assume that if a compromised host is used, the adversary can simply take screenshots for all transactions performed.

B. Scheme Limitations

The scheme is limited to not connecting a secure device to the possibly compromised host; the e.dentifier2 provides maximum security when connected to the host. Moreover, no software, other than what exists in most computers should be installed. Last, if the customer enters data twice, once in the keyboard of the host and once in the secure device, there can be high security. The proposed scheme is also limited to not requiring the user to enter data twice.

C. Point of Implementation

There are three points in the transaction procedure for the one - time application to be implemented. The first option is to try to simulate the "See What You Sign" principle as found in the e.dentifier2 connected - mode. In this case we need to build a one - time application for the last part of the transaction; the application would take over the transaction for messages 4 and 5, as seen in Figure 1. The goal of such an application would be to make the transaction receipt harder to seem legitimate in case of fraud. In such a case, the customer would have to notice being under attack and cancel the transaction. The second option is to perform the whole transaction, including the login process, through the application. Compiling and signing one - time applications is an operation that requires resources on the server side; this would make it possible for Internet criminals to perform Denial of Service attacks. The third option is to use the application for the transaction after the login process. This way it is possible to make it harder for the attacker to enter the fraudulent data in the first place, while we also make the transaction receipt harder to manipulate.

D. Runtime Environment

One important decision that needs to be made, is whether the application will run on a browser or independently. Both options are viable, however customers will probably be reluctant to perform the beginning of the transaction on a browser and finalize it on another program. Running a Java applet within the browser fits this criterion. Moreover, Java applets run within a Virtual Machine; this provides for code isolation. Last, API hooking does not apply to Java applets, at least not in the same sense as with other applications. Adversaries would have to perform the attack on a lower level, which means that the attack can be more expensive.

E. Code Signing

Code Signing is essential in order to provide proof of integrity and authenticity of the application in use. The Public Key Infrastructure (PKI) is used to provide this proof. One major problem is that criminals can insert fraudulent root certificates to OSs. This way code that is not legitimate may appear to be signed by a trusted source. Furthermore, users do not always pay attention to certificates; they will underestimate the warnings provided by their browser if a website is not trusted. Users also underestimate the risks related to using an application that is not signed.

Related to Java PKI, Java applets can be signed by a trusted source. Moreover, there are features that would terminate the application if code is modified or if unsigned code is used along with signed code. However, there is limited research on the level of effectiveness of these features. Most Java - related security research is about ensuring that the host will not suffer from a malicious applet. There are indications that Java security can be completely shut down [13]. This means that although the application should be signed, the security of the scheme cannot solely depend on the possibility that customers check the certificate of the application properly.

F. Local Code

One of the points of attack for MitB is API hooking. One of the solutions proposed is Hardened Browsers. Hardened Browsers disallow extensions, have Javascript disabled, are statically compiled and use local code as much as possible [8]. Through these measures they have managed to disallow MitB attack, at least for now. One important disadvantage is that banks cannot enforce customers to use hardened browsers. Moreover, such solutions will not work in environments, over which users

have limited control. However, the concepts that they use for increased security can be used in other schemes. The code of the application should be as local as possible. Using libraries provided by the underlying OS should be avoided if possible, since libraries can be compromised and introduce points of attack [14].

1. Encryption

The application to be implemented should not rely on SSL/TLS libraries provided by the underlying OS. The application should encrypt data by itself, so that there are fewer opportunities to manipulate cleartext transaction details. This can increase the level of security, however kernel - mode rootkits are possibly capable of extracting the key from the memory and manipulating data in the memory level. Obfuscating memory entries in such low level is also possible, but possibly not effective.

G. Code Obfuscation

Code obfuscation makes it harder for an adversary to analyse the code, yet not impossible. If criminals put effort on it, eventually they will be able to reverse engineer the application and find ways to attack. On the other hand, if variants of the application with randomization features are used for each transaction, then it will take more effort on the attacker's side.

H. Application Randomization

Certain concepts can increase the security of the application. Using functions only once can increase the level of security, by making it harder for the attackers to find patterns within the code. In our application, the main target of randomization is to make the transaction receipt harder to manipulate; the adversary should not be aware of which methods to intercept in order to display the information he wants without obvious effects to the legitimate user. One way to achieve this is by breaking strings into characters. This way, regular expressions will not work efficiently; attack will be harder to automate. For increased security we can change numbers into the words, i.e. "1" into "one". This would provide a larger set of characters and confuse attackers even more. "One" in turn should consist of three textboxes, one for each character. Each textbox could be displayed to its unique coordinates by its own method. The order in which these methods are called can be randomized. This way, criminals will have a harder task finding which textboxes they need to manipulate. It should be noticed that measures, like changing "1" into "one" may decrease usability. Not only users will have a harder time to read the text, but it is also possible that some customers will not know how to read these words, while numbers are practically universal.

I. Data Input

The way that users input data for their transactions has a major effect on the levels of security and usability. If the application only simulates the "See What You Sign" principle, then the user will input the transaction details as normal. The response to the challenge (message 5 in Figure 1) can be entered through the keyboard or by using graphical keyboards. If the whole transaction is performed through the application, the user may input all data through graphical keyboards or through the hardware keyboard. The level of usability and security can be affected by this decision. In both cases input is vulnerable to manipulation, but generating valid messages by automated manipulation of mouse events is a harder task for the adversary. The side - effect is that it is also less usable for most legitimate users.

J. Pushing Code

One important trait of a one - time application is that it is supposed to be used only once. Every time a bank customer wants to perform a transaction, he will have to download a new application. This means that every time there is a security breach, the application can be updated in a way that makes it secure again. The user does not need to allow for an update in his application, it will happen automatically since he will be downloading a new one anyway. On the other hand, there may be vulnerabilities in the JRE. Unless the JRE is updated by the user, adversaries will be able to take advantage of these vulnerabilities.

VI. RESIDUAL RISKS

The one time application can increase the security of IB transactions, but there are still attacks that can be successful. The threat of MitM attacks remains. The application can improve security against content manipulation attacks, but if adversaries successfully pass an overlay application to the user, they can still perform fraudulent transactions.

There are two ways to defend against such an attack. Unfortunately, both will also decrease the level of usability. The first way is through the use of graphical keyboards for data input. This will work under the assumption that adversaries will not be able to automate data input;

they will not know where they need to click to perform the transaction if the layout of the application is randomized. Moreover, clicking on e.g. button "1" does not necessarily mean that the message sent to the bank server will include "1". Messages can be encoded, so that adversaries do not know which messages to send, unless they click on the button. The attack will be harder to automate, but a criminal sitting in front of his computer will still be able to attack this scheme.

The second way to defend against such an attack is by making it harder for the attacker to extract the challenge from the legitimate applet. The challenge should be obvious to the legitimate user, so if the adversary sits behind a monitor, he will also be able to extract the challenge and pass it to the user. However this is not an automated attack. Currently malware uses regular expressions in order to find which data need to be manipulated. The efficiency of regular expressions can be reduced by breaking strings into characters and displaying them in a randomized order. This can be done transparently to the user by positioning the characters in the correct coordinates, but displaying them in a different order than normal. A vulnerability of this is that there is already software that can break CAPTCHAs. Malware that has similar functionality will be able to extract the challenge from the application.

VII. JAVA CONSIDERATIONS

In most cases, users have Java installed on their computers. In this case, users do not need to install software to run the application. However, if the JRE does not exist on a host, customers will need to install it; this requires administrative privileges. Moreover, a number of security aware users do not allow Java applets to run on their host, since they are afraid of malicious applets. Last, there are popular devices that do not support Java at all, e.g. the iPad. Although there are IB applications for these devices, customers may still want to use IB in the same way as in their desktops or laptops.

There are security considerations related to Java. Java applets run in a sandbox, which in theory would keep the underlying host intact, even if the code is malicious. However, sandboxes are not always effective. Thus, users do not always wish to allow Java applets to run. Moreover, earlier versions of the JRE would provide higher privileges to trusted applets, i.e. applets signed by a trusted source. This lead some users to disallow signed Java applets to run on their computer.

VIII. CONCLUSIONS AND FURTHER RESEARCH

Using one - time applications for IB transactions can mitigate the risks related to the MitB attack, as it is performed at the moment. Moreover, it can do it in a usable way. It should be expected that new types of attacks will arise against the application, if it is popular enough. One can think of this as the next stage in the Cat and Mouse game. Since adversaries have full power over compromised hosts, securing the transaction against all types of attacks may be impossible without a trusted display.

Making the application totally secure is not the real target. Finding the balance between security and usability is more important from a marketing perspective. Scenarios like the overlay application will succeed if the adversary sits in front of his computer. One realistic goal is to make it as hard as possible for the attack to be automated. Automation may still be possible, but if the application changes frequently, it is possible that Internet criminals stop targeting it. The security of the application relies on its obscurity and on the fact that it is easy to update. One question is how long it will take, until Internet criminals find a generic way to perform the attack, in spite of the obfuscation.

The functionality of the application depends on the balance between security and usability that needs to be defined. From a security perspective, performing the whole transaction through the application may increase the security level, while the scheme remains usable. Usability is an important factor of the equation for the ideal IB scheme. e.dentifier2 connected - mode is known to be secure; one idea would be that ABN Amro forces all customers to use that for IB. However, this would probably drive customers away. Using graphical keyboards may increase the level of security but it will also decrease usability. Other measures to confuse the criminals can be deployed, but it is quite probable that they will also confuse the legitimate users to some extent. Thus there is need for a usability survey that will show what customers are willing to do for secure IB transactions. Moreover, the scheme proposed should be implemented and penetration testing should be performed before the scheme is deployed.

The application will be pushed to the bank customers, thus there will never be an outdated application in use. This does not apply for the JRE, which may be vulnerable; JRE will not be updated automatically. There are certain shortcomings related to the use of Java. A lot of customers do not have it installed in their computers, while others have it disabled. Even worse, popular devices such as Ipad do not support it at all. Moreover, Java security tries to make sure that malicious applets do not harm a host. There is little research done on how secure the security features of Java actually are. If the same concepts can apply to other runtime environ-

ments, complying with the same requirements, then the security and usability of those environments need to be researched.

Last, building and signing applications at real time is an expensive operation. Customers will need more time to perform transactions. On the server side, the operations require increased resources. If the scheme is secure enough, implementing it is within the capabilities of the bank, however the scalability of the scheme needs to be investigated.

IX. ACKNOWLEDGEMENTS

I would like to thank my supervisor Sander Vos, Steven Raspe and Han Sahin for all their help during the project.

[1] P. Hanacek, K. Malinka, and J. Schafer. e-banking security - a comparative study. *Aerospace and Electronic Systems Magazine, IEEE*, 25(1):29 –34, jan. 2010.

[2] R. Oppliger, R. Rytz, and T. Holderegger. Internet banking: Client-side attacks and protection mechanisms. *Computer*, 42(6):27 –33, june 2009.

[3] Morten Hertzum, Niels Jrgensen, and Mie Nrgaard. Usable security and e-banking: ease of use vis-a-vis security. *Australasian Journal of Information Systems*, 11(2), 2007.

[4] Fred Douglis. `Phone + Internet Cafe = Secure Banking? You Betcha`. *Internet Computing, IEEE*, 13(6):4 –5, nov.-dec. 2009.

[5] The Open Web Application Security Project (OWASP). Man-in-the-browser attack. `https://www.owasp.org/index.php/Man-in-the-browser_attack`.

[6] Thomas Weigold, Thorsten Kramp, Reto Hermann, Frank Hring, Peter Buhler, and Michael Baentsch. The zurich trusted information channel an efficient defence against man-in-the-middle and malicious software attacks. In Peter Lipp, Ahmad-Reza Sadeghi, and Klaus-Michael Koch, editors, *Trusted Computing - Challenges and Applications*, volume 4968 of *Lecture Notes in Computer Science*, pages 75–91. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-68979-9_6.

[7] Catherine S. Weir, Gary Douglas, Martin Carruthers, and Mervyn Jack. User perceptions of security, convenience and usability for ebanking authentication tokens. *Computers and Security*, 28(12):47 – 62, 2009.

[8] Philipp Guhring. Concepts against Man-in-the-Browser Attacks. 2006.

[9] R. Oppliger and R. Rytz. Does trusted computing remedy computer security problems? *Security Privacy, IEEE*, 3(2):16 – 19, march-april 2005.

[10] Mohammed AlZomai, Bander AlFayyadh, Audun Jøsang, and Adrian McCullagh. An exprimental investigation of the usability of transaction authorization in online bank security systems. In *Proceedings of the sixth Australasian conference on Information security - Volume 81*, AISC '08, pages 65–73, Darlinghurst, Australia, Australia, 2008. Australian Computer Society, Inc.

[11] Symantec. Windows rootkit overview. `https://www.symantec.com/avcenter/reference/windows.rootkit.overview.pdf`.

[12] Charlie Kaufman, Radia Perlman, and Mike Speciner. *Network Security: Private Communication in a Public World, Second Edition*. in computer networking and destrilanted systems. Prentice Hall PTR, April 2002.

[13] A. Dabirsiaghi. Javasnoop: How to hack anything in java. BlackHat Las Vegas, 2010.

[14] J. Berdajs and Z. Bosni. Extending applications using an advanced approach to dll injection and api hooking. *Software: Practice and Experience*, 40(7):567–584, 2010.