Chapter 1

SECURITY TESTING OF THE ONLINE BANKING SERVICE OF A LARGE INTERNATIONAL BANK

Andre L.M. dos Santos, Giovanni Vigna, Richard A. Kemmerer
andre@cs.ucsb.edu, vigna@cs.ucsb.edu, kemm@cs.ucsb.edu
Reliable Software Group
Department of Computer Science
University of California, Santa Barbara

Keywords: Security, Online Banking, Tiger Teams, Penetration Testing, Social Engineering

Abstract

Online banking and electronic commerce have become an everyday reality for millions of users. Almost every large banking institution offers services such as account management, fund transfers, automatic payments, and investments through the Internet. The quality of the provided services has become a driving factor in user selection of a banking institution. Given the critical nature of the services provided, banks and financial institutions are investing substantial resources in the implementation of sophisticated financial applications that are appealing to the end-user. In the design and implementation of these applications developers face a trade-off between user-friendliness and security.

This paper presents the results of the security testing of the online banking services of a large international bank. The testing process followed a purely black-box approach. Starting from a single legitimate account and unprivileged Internet access to the bank's site it was possible to compromise the security of many accounts. The results of the study show how user-friendliness may clash with security requirements leading to critical flaws in the system. The lessons learned from this experiment represent an initial step towards a more careful design of online financial applications.

Introduction

Today, millions of users access financial services through the Internet (Ghosh, 1998). Online banking offers many advantages to the banks, which reduce their operating costs and reach a larger portion of their clientele, and to the users, which receive 24 hours a day/7days a week banking services from their homes and workplaces.

The services offered range from simple account management to fund transfers and online investments. The volume of the online transactions performed may vary from a few dollars to thousands of dollars, but the overall volume of the transactions is large and is destined to become a substantial portion of the banking services. In this scenario, banks and financial institutions struggle to captivate users through the provision of sophisticated services and appealing, user-friendly applications.

These applications suffer from the problems of any Internet- and World Wide Web-based application. The large-scale distributed nature of the application architecture, the use of an insecure communication infrastructure, such as the Internet, the use of WWW technology, which historically suffers from many security flaws (Paoli et al., 1998; Dean et al., 1996), and the requirements of the users in terms of usability make the design of a secure online banking service a difficult task. In addition, the reduced time-to-market and the need to reach a wide range of users put tremendous pressure on the application developers, sometimes leading to insecure implementations. In the past, flaws in online banking applications have been exploited by hackers to compromise the security of user accounts. These attacks are seldom publicized because of the impact that they may have on users' confidence in the compromised financial institutions. Yet, it is important to evaluate how flaws may appear in these systems in an effort to develop a set of methodologies, techniques, and guidelines that may help the application developer to deliver secure and user-friendly online banking applications.

This paper describes the security testing performed against the online banking services of Bank X, a large international bank with more than 30 million accounts and approximately 400,000 online accounts. Security testing (also known as "penetration testing") aims at determining weaknesses in the security protections of a system. There are very few documented experiments like the one described in this paper. Banks are usually not happy to have their security flaws exposed. In addition, the testers often don't want to discuss their approaches and techniques.

The experiment is interesting also because it shows an example of the trade-offs between security and user-friendliness. As it will be explained later, the adoption of user-defined personal identification numbers (PINs) and meaningful error messages disclosed security-critical information during the attack. In addition, this case study shows how vulnerable these systems are to social engineering attacks.

The remainder of this paper is structured as follows. Section 1 describes the approach that was followed in the testing process. Section 2 describes the architecture of the on-line system under attack. Section 3 details the actual testing process. Finally, section 4 draws some conclusions and discusses the lessons learned from this experiment.

1. THE APPROACH

The goal of this security testing process was to compromise one or more accounts using the online banking service. Compromise includes access to private information (e.g., access to the account balance) and the ability to transfer funds from the attacked account to another.

The approach followed is pure *black-box testing.* That is, no information other than the name of the web site providing the service was known prior to the beginning of the attacks. We acted as "hackers" interested in penetrating or circumventing the security protections of Bank X's network-based services. However, our approach differed from a real attack in two ways. First, we took extra precautions not to jeopardize the mission of Bank X. This of course would not be a concern if we were malicious hackers. For example, some attacks that may have disrupted the infrastructure were performed during a well-defined time window that was determined by interacting with the security officers of Bank X. Second, we did not use any caution in hiding our attempts to probe for and then exercise security vulnerabilities in the online banking system. This was done to test the ability of Bank X to detect and record even the most blatant attack signatures.

At the beginning of the study we had no idea what the protocols used by the applications were, what the account number format was, or even what the account lockout procedure was. Before starting the attacks we opened a legitimate account at one of Bank X's branches. The account was accessible online, and it helped us to determine what protocols and message formats were used.

The study was a several month effort that began in November 1998 and ended in January 1999. The study revealed weaknesses in the online system and identified vulnerable areas that resulted in a major redesign of the system. This document is the first authorized disclosure of the attack against Bank X.

2. THE ONLINE BANKING SYSTEM

Bank X's online banking system is based on WWW technology. A user connects to a web server using a web browser. TCP/IP connections between the client and the server are protected against eavesdropping by means of the SSL protocol (Freier et al., 1996).

The user begins an online banking session by filling in four text fields of an applet or HTML form. These fields are: branch number, account number, control digit, and PIN. The sizes of these fields are:

- branch number: maximum of 4 digits;
- account number: maximum of 6 digits;
- control digit: 1 digit;
- PIN: 4 digits.

Bank X is very concerned about their online banking system being user-friendly. In order to achieve this goal, specific information is returned to the user in case one or more of the fields are filled with incorrect information. In addition, this is done in an ordered and deterministic way. That is, first the branch number is checked. If the check fails, the system will generate an answer saying that the branch number is incorrect. Then the application checks if the control digit is correct with respect to the account number. If it is not so, a message saying that the control digit is incorrect is returned. If the control digit is correct for the given account number, then the account is checked. The account may be non-existent or not registered for Internet access. In each case, a different answer is returned. Finally, the system generates a specific answer for an incorrect PIN.

The system has an additional authentication procedure. After filling the fields with correct information another web page is presented to the user. This page has one or two HTML form fields depending on whether it is a business account or a personal account. These fields are used to request personal information, such as SSN, EIN, or mother's maiden name. The system logs the user in only after checking the correctness of this information.

Thus, in order for a user to log into his/her account, he/she needs to give correct answers for the challenges of two different web pages. That is, the user needs to enter a correct branch number, account number, control digit and PIN in the first web page, and after this the user needs to fill the correct personal information, related to the specific account in a second web page. After the information has been validated by the server, the user is presented with a web page that gives the ability to check balance, perform transfers, change personal information, and pay bills.

3. THE ATTACK

The attack was carried out through a series of steps. In the following sections the steps are presented sequentially, but the actual testing sometimes required backtracking to previous steps.

3.1. UNDERSTANDING THE SERVICES

The first step of the attack was to leverage off of the legitimate account under our control to determine the characteristics of the service, such as its general structure and the lockout policy.

The online banking system locks out after entering either three wrong PINs or two wrong personal data items. The lock out happens even if the second wrong personal data is entered after the attacker enters one wrong personal data and exits the system; i.e., the system maintains status information across sessions. The experiments also determined that the lock outs are reset every day at midnight.

During this phase our efforts were concentrated on reverse engineering of the applet used on the client side. The bank's application was a Java applet that used three Java classes for account authentication. These classes had names that were random and unique for each page hit. This protected the application against an attack where a malicious user infiltrates a Trojan horse Java class in order to get control of a site's applet (dos Santos, 1997).

The bank applet's Java classes were obfuscated in order to make it harder for someone to disassemble or decompile them. As a result, the Javasoft JDK disassembler was not able to disassemble these classes, and the Mocha and Jasmine decompilers were not able to decompile them. In order to fix this we developed a pre-decompiler. This pre-decompiler cleans up the classes' bytecode by modifying two patterns that were discovered not to be properly formatted.

Both patterns were related to the number used for the size of the parameters inside the class file. According to the Java virtual machine specification (Lindholm and Yellin, 1999) many parameters in a Java class file have variable size and as such must have their size in bytes explicitly specified. The first obfuscation was accomplished by declaring strings in the "constant_pool" array of a class file to be of a bigger size than they really were. This was identified by the pre-compiler by searching for the null byte (`00`) at the end of the string. This byte belonged to the next declaration in the "constant_pool" and indicated the real end of the string. The other obfuscation was accomplished by inflating a line feed byte (`0A`) to be a line feed plus carriage return byte (`0A 0D`) without updating the size of the parameter. This meant that the parameter was declared with a smaller size than it really was. This was fixed by collapsing any combination of line feed plus carriage return into a line feed byte.

The three obfuscated classes used for account authentication were decompiled by Jasmine after being processed by the pre-decompiler. These classes provided the functionality for a user interface, a crypto algorithm, and an interface to the crypto algorithm. The crypto algorithm was not studied to verify its correctness or security.

3.2. DEVELOPMENT OF A CUSTOM APPLICATION

The successful reverse engineering of the client side applet allowed us to develop a Java application that interacted directly with Bank X's site. The application used two of the three original classes with very little modification. The Bank X applet, and therefore our application, uses a timestamp that is supposed to control how long a request may live. The timestamp is obtained by performing a get operation to a specific URL. This information is provided for any request and can be acquired every time the application needs to encode information to be sent to the server. This solution was not attractive, however, due to the network delay between requests and replies. Instead, the local clock was synchronized with the bank clock at start time.

The four text fields (branch number, account number, control digit, and PIN) are encrypted together with the timestamp by the crypto class. An HTTP `GET` request is performed by the Java applet after converting the encrypted information to a string. The return value of the `GET` request is a nonce. The nonce is then used to perform a second `GET` request that returns a page, which is shown in the user's browser where the original applet is running. This page either requests personal information data or reports that some of the fields were filled incorrectly. The applet uses the `showDocument` method to display this page. Our Java application, on the other hand, is only interested in parsing this page to get the values returned. Therefore, the application stops after receiving enough of the page contents to allow it to identify the answer received.

It should be noted that the nonce can be used only once (as it should be). This means that if an attacker hijacked or fabricated a nonce, then there is a chance that he could use it before the legitimate user does. This would allow the attacker to fetch the second page of the online banking system, which requests personal information. The nonce by itself does not give away information about the account number for which it was generated, but the page that is fetched using the nonce implicitly has this information. Thus, an attacker could possibly gain access to a user's account with a good database of personal information and a valid nonce. The attacker could also use the lock out procedure to deny service to the rightful user while and after performing an attack that hijacks a nonce.

3.3. DETERMINING VALID BRANCH AND ACCOUNT NUMBERS

The experiments performed on the online system attempted to find out what accounts existed, whether they were Internet-accessible, what the PIN number for each account was, who owned the accounts, and eventually gain complete access to the accounts on the online system. A series of experiments were performed to see how difficult it would be to get any or all of this information. These experiments were performed by using Bank X's online banking services at Bank X's site, as well as by social engineering. The following subsections describe the experiments and their success.

The first experiment that was run attempted to demonstrate that the valid accounts for a particular branch could be readily determined, but this required first determining the valid branch numbers. Finding branch numbers is relatively easy, since branch numbers for all Bank X branches are available at Bank X's web site. Thus, an attacker can select the branches that are most interesting (e.g., those in large cities) and attack them.

As already mentioned, account numbers are composed of a six digit number and an additional control digit. When we first started this experiment we tried all ten possible control digits for each account number tried. Using this approach it was necessary to test an average of five control digits before finding the correct one. For each attempt an answer saying "bad control digit" indicated that a wrong control digit was tried. After collecting more than 300 correct account numbers and corresponding control digits we were able to correctly reverse engineer the algorithm that is used to generate the control digit.

We do not know if the algorithm used for the generation of this control digit is in the public domain, but because it was not available to us we were forced to determine it by reverse engineering. By using the control digit generation algorithm, answers saying that the control digit or branch number are incorrect were eliminated. The remaining answers enabled us to map a branch with regard to accounts that were in use and those that were accessible through the Internet. As a bonus, if an account used the PIN that was tried, this was reported too.

3.4. DETERMINING VALID PINS

Many online banking services rely on a fixed length personal identification number (PIN) to identify a user. Some banks, allow access to all of their online operations after a successful PIN is entered; others require additional identification, like social security number, mother's maiden name or an additional PIN.

As with passwords, users have difficulty in remembering large personal identification numbers. Therefore, there is a natural tendency to use small, easy to remember numbers (like birthday or 1234). Anticipating the problems that this class of numbers can represent, many Internet banking applications require users to choose PINs that are not easy to guess. In the interest of user-friendliness, however, the banks cannot require the user to use, and remember, a very large number. Therefore, it is a wide spread practice to use 4 or 6 digit PINs. Unfortunately, because of the small size of the PINs, an attacker can target a particular account and try all possibilities. In order to defend against this class of attacks, banks usually lock out accounts after a certain number of unsuccessful identification attempts.

The success of our attack relied on the ratio between the size of the personal identification number and the number of users of the service. If we were to fix the account and vary the possible PINs this would cause a lock out in the particular account after a minimal number of failures. Instead, we fixed the PIN and varied the account number. This resulted in no lock outs, since a particular account will be tried again with a different PIN only after numerous other accounts have been tried. Thus, the lock out protection is not triggered. Some banks, however, rely on an additional IP-triggered protection that locks out specific IP addresses after a certain number of failed attempts from the same IP address. This can be bypassed by means of IP spoofing (Bellovin, 1990). Bank X used this additional lock out mechanism.

Because the bank PIN numbers are only four digits long, there are only 10,000 possible PIN values. This small number of PIN values coupled with the large number of online accounts (390,000) makes a random PIN guessing attack very attractive to the attacker. If the PIN numbers were uniformly distributed and the PIN number guess was randomly generated, then the guess would be successful in matching the correct PIN for one in every 10,000 accounts tried (i.e., .01%). Thus, 39 accounts would have their PIN compromised for every PIN number guessed. Unfortunately, the common use of easy to guess PINs makes the system even more vulnerable. When we ran the first experiment, which mapped accounts for each branch, we used the PIN 1234. This experiment revealed that 3% of the online users from the test branches were using PIN 1234.

When running the account mapping experiment it was discovered that the transaction speed using the Internet at the test site varied significantly. This could be viewed as an advantage for the bank in discouraging attackers. We found, however, that even with this variation the average time required to test one account was only six seconds. Using this average and noting that account numbers can go up to 999999, it would take 6,000,000 seconds or about 70 days to test all possible accounts at a single branch. This time could be reduced significantly, however, by performing parallel accesses. When running the experiments we determined that the delay was mostly caused by the reduced bandwidth and not by server saturation. To verify this, three applications were run in parallel and it was observed that the throughput for each application had not changed significantly from the throughput observed when only one application was running.

Another observation that was made when mapping the valid accounts was that the branches give preference to low numbered accounts. As a result, an attacker could successfully map a large portion of the valid accounts by starting with account 1 and trying account numbers in order up to a prefixed value. We found that trying values lower than 100,000 gave good results in the experiments.

3.5. FINDING ACCOUNT OWNERS

After determining the valid accounts with online banking access and their associated PINs, the next step was to determine the name of the account owner. This is important information for the attacker, and it is sensitive information to the bank. One might think that it would be difficult for the attacker to find out this information. Our experimentation showed that with access to one legitimate account it is trivial to get all account owners of valid accounts in a branch, with very few exceptions.

To find account owners an attacker with access to an account only needs to use the transfer facility of the online banking system. To be more specific, the attacker initiates a transfer from his/her account to the account for which he/she wants to know the owner. This action causes the server to return a page that contains the name of the owner of the account and that asks for confirmation of the transfer. Now that the attacker has the desired information, he/she can record it and cancel the transfer operation. In a few cases this approach wasn't successful because the account was either "blocked" or "non-authorized".

To demonstrate the feasibility of this approach for identifying the owner of an account a program was developed that repetitively performed the operation of initiating a transfer and aborting it. The program generated transfer requests using a valid login to the system using the account to which we had legitimate access. One inconvenience that had to be overcome was that a session in the online banking system has a fixed lifetime. That is, the session is no longer valid after some time. To overcome this the program recognizes when the lifetime expires, and, when necessary, it logs into the system again starting a new session.

3.6. FINDING PERSONAL INFORMATION

The final step to achieve complete access to the accounts is to answer one or more questions about the owner of the account. More specifically, the applet embedded in the first page of the online banking system, which is emulated by the developed application, returns a nonce. This nonce is then used to construct a URL that is used to fetch an HTML page. In the case where the correct PIN number was entered, this page requests information based on the account owner's personal data. On this page it is explicitly stated that the personal data is randomly selected. When running the experiments we found that for personal accounts the system requested two different pieces of data each time, and these were requested in a non-deterministic way. The two requests were chosen from SSN, date of birth, father's name, and mother's maiden name. For company accounts the information requested was always the company's tax identification number, EIN.

To obtain information about personal account holders we first decided to use social engineering. We assumed that we would have a better chance of obtaining the required information for a branch in a small town where we had access to people who might know personal information about the individual that owned the account. Thus, to test our assumption we focused on a branch from a small town and repeated the experiments presented above. In a matter of a few hours we were able to obtain the PINs for 25 accounts at that branch. Using the name of the owner of one of these accounts we were able to obtain the required personal information by asking questions of someone in the town. This enabled us to completely compromise our first account.

We were not completely satisfied with the approach since it required the attacker to obtain the personal data by social engineering, which could be very time consuming. In addition, this type of social engineering adds more risk for a potential attacker. Therefore, we decided to investigate how difficult it could be to get the personal information that is needed from publicly available data sources outside the bank system. We found that we were able to call the governmental department in charge of distributing SSNs and EINs, and get these numbers. To demonstrate the feasibility of this approach, we used this method to compromise two personal accounts and two business accounts. One of the business accounts was for a well-known foreign corporation.

In analyzing how personal information is requested by the online system and how the answers are returned, we discovered some critical weaknesses in the approach. In particular, we discovered that we were able to choose which personal information to return to the system, regardless of what information was requested. To be more specific, the server does not keep track of what questions were asked. Instead, it uses the text field name in the page to determine what was requested. Therefore, by changing the text field appropriately, we were able to answer the questions that we knew the answer to, rather than the questions that we were asked[1]. Thus, an account could be compromised even if only some of the information that could be requested was known. A more thorough analysis of this part of the system revealed that for business accounts the personal information requests could be completely bypassed. Recall that for business accounts the EIN was always requested. It is not difficult to find out the EIN of a company, but we found an even easier approach. When asked for an EIN we again changed the field name in our coded response. However, instead of giving the answer to the question asked we gave an answer stating that the name of the father is "`null`". Since business accounts never have a father, the system validated this answer and logged us into the company's account. This turned out to be particularly useful because we had previously found that for some special accounts we were unable to get the name of the business or person that owned the account. Our routine identified these accounts as "non-authorized". For these accounts it was impossible to get the necessary personal data using the publicly available data sources outside the bank. However, with the bypass method, this was no longer necessary. To demonstrate the feasibility of this approach, we used the bypass method to compromise the account of a large multinational corporation, which regularly performed transfers of \$100,000 or more.

4. CONCLUSIONS

The security testing performed against Bank X's online banking system was successful in compromising a number of accounts. We developed an approach to collect valid account numbers and PINs and a second approach to automatically associate these accounts with the person or business that owned the account. This information could then be used to obtain personal information from publicly available data sources outside of the bank system, which gave us complete access to the accounts. In addition, we developed an approach that bypassed the personal information requests for business accounts, which allowed us to log into these accounts. Once this was done we not only knew who owned the accounts, but we also got all of the other online capabilities for the accounts.

All experiments in this study were conducted using a Java application that disguised itself as the bank's applet. Therefore, no particular exploits using HTML forms were studied. However, most or all of the attacks described for the applet would also work with HTML forms. It is also possible to explore vulnerabilities that are particular to HTML forms, such as JavaScript attacks in older browsers. This access redundancy, which again was done for user-friendliness, is another weakness of the system, since it enables an attacker to explore flaws found, or to be found, in both HTML forms and Java applets. The attacker can then attack the subset of the clients that use that type of access.

This experiment made the tradeoff between user-friendliness and security very clear. In an attempt to give informative answers to users the application disclosed security relevant information. This flaw was present at several points in the banking application. Banking application developers should be very careful in deciding the type of diagnostic messages returned to the users in case of errors. Messages should be informative but should also be generic enough not to allow an attacker to determine what part of a set of submitted data is the cause of the error.

This paper did not discuss possible denial of service attacks, either against the banking application (e.g., lockouts, flooding), or against the communication infrastructure (e.g., crashing of essential services, distributed denial of service). Recent attacks against major Internet sites (Dittrich, 1999) showed that these attacks are very difficult to block and that application designers should take into consideration the possibility of service disruptions.

Notes

1. This is a variation of the well known value checked vs. value used security problem (Bisbey et al., 1975).

References

Bellovin, S. (1990). Security Problems in the TCP/IP Protocol Suite. *Computer Communications Review*, 19(2).

Bisbey, R., Popek, G., and Carlstadt, J. (1975). Inconsistency of a Single Data Value Over Time. Technical Report ISI/SR-75-4, USC Information Sciences Institute.

Dean, D., Felten, E., and Wallach, D. (1996). Security: From HotJava to Netscape and Beyond. In *Proceedings of the IEEE Symposium on Security and Privacy.* http://www.cs.princeton.edu/sip/pub/secure96.html.

Dittrich, D. (1999). The DoS Project's "trinoo" distributed denial of service attack tool. http://staff.washington.edu/dittrich/misc/ddos/.

dos Santos, A. (1997). Another way to exploit local classes in Java. Risks 19.41.

Freier, A., Karlton, P., and Kocher, P. (1996). The SSL Protocol Version 3.0. INTERNET-DRAFT.

Ghosh, A. K. (1998). *E-Commerce Security: Weak Links, Best Defenses.* John Wiley and Sons.

Lindholm, T. and Yellin, F. (1999). *The Java Virtual Machine Specification.* Addison-Wesley, 2nd edition.

Paoli, F. D., dos Santos, A., and Kemmerer, R. (1998). *Web Browsers and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 235–256. Springer-Verlag.