Secure Input for Web Applications

Martin Szydlowski, Christopher Kruegel, Engin Kirda
Secure Systems Lab
Technical University Vienna
Vienna, Austria
{msz,chris,ek}@seclab.tuwien.ac.at

Abstract

The web is an indispensable part of our lives. Every day, millions of users purchase items, transfer money, retrieve information and communicate over the web. Although the web is convenient for many users because it provides anytime, anywhere access to information and services, at the same time, it has also become a prime target for miscreants who attack unsuspecting web users with the aim of making an easy profit. The last years have shown a significant rise in the number of web-based attacks, highlighting the importance of techniques and tools for increasing the security of web applications.

An important web security research problem is how to enable a user on an untrusted platform (e.g., a computer that has been compromised by malware) to securely transmit information to a web application. Solutions that have been proposed to date are mostly hardware-based and require (often expensive) peripheral devices such as smartcard readers and chip cards. In this paper, we discuss some common aspects of client-side attacks (e.g., Trojan horses) against web applications and present two simple techniques that can be used by web applications to enable secure user input. We also conducted two usability studies to examine whether the techniques that we propose are feasible.

1 Introduction

Since the advent of the web, our lives have changed irreversibly. Web applications have quickly become the most dominant way to provide access to online services. For many users, the web is easy to use and convenient because it provides anytime, anywhere access to information and services. Today, a significant amount of business is conducted over the web, and millions of web users purchase items, transfer money, retrieve information and communicate via web applications.

Unfortunately, the success of the web and the lack of technical sophistication and understanding of many web users have also attracted miscreants who aim to make easy financial profits. The attacks these people have been been launching range from simple social engineering attempts (e.g., using phishing sites) to more sophisticated attacks that involve the installation of Trojan horses on client machines (e.g., by exploiting vulnerabilities in browsers in so-called *drive-by attacks* [19]).

An important web security research problem is how to effectively enable a user who is running a client on an untrusted platform (i.e., a platform that may be under the control of an attacker) to securely communicate with a web application. More precisely, can we ensure the *confidentiality* and *integrity* of sensitive data that the user sends to the web application *even if* the user's platform is compromised by an attacker? Clearly, this is an important, but difficult problem.

Ensuring secure input to web applications is especially relevant for online services such as banking applications where users perform money transfers and access sensitive information such as credit card numbers. Although the communication between the web client and the web application is typically encrypted using technologies such as Transport Layer Security [9] (TLS) to thwart sniffing and man-in-the-middle attacks, the web client is the weakest point in the chain of communication. This is because it runs on an untrusted platform, and thus, it is vulnerable to client-side attacks that are launched locally on the user's machine. For example, a Trojan horse can install itself as a browser-plugin and then easily access, control, and manipulate all sensitive information that flows through the browser.

Malware that manipulates bank transactions already appears in the wild. This year, for example, several Austrian banks were explicitly targeted by Trojan horses that were used by miscreants to perform illegal money transactions [13, 21]. In most cases, the victims did not suspect anything, and the resulting financial losses were significant. Note that even though the costs of such an attack are covered by insurance companies, it can still easily harm the public image of the targeted organization.

A number of solutions have been proposed to date to enable secure input on untrusted platforms for web-based applications. The majority of these solutions are hardware-based and require integrated or external peripheral devices such as smart-card readers [10, 23] or mobile phones [15]. Such hardware-based solutions have several disadvantages. They impose a financial and organizational burden on users and on service providers, they eliminate the anytime, anywhere advantage of web applications and they often depend on the integrity of underlying software components which may be replaced with tampered versions [12, 24, 25].

In this paper, we discuss some common aspects of client-side attacks against web applications and present two simple techniques that can be used by web applications to enable secure input, at least for a limited quantity of sensitive information (such as financial transaction data). The main advantage of our solutions is that they do not require any installation or configuration on the user's machine. Additionally, in order to evaluate the feasibility of our techniques for mainstream deployment, we conducted usability studies. The main contributions of this paper are as follows:

- We present a technique that extends graphical input with CAPTCHAs [3] to protect the *confidentiality* and *integrity* of the user input even when the user platform is under the control of an *automated* attack program (such as a Trojan horse).

- We present a technique that makes use of confirmation tokens that are bound to the sensitive information that the user wants to transmit. This technique helps to protect the *integrity* of the user input even when the user platform is under the control of the attacker.

- We present usability studies that demonstrate that the two techniques we propose in this paper are feasible in practice.

This paper is structured as follows: Section 2 gives an example of a typical client-side attack. Section 3 presents our techniques to enable secure input for web applications. Section 4 presents the results of our user studies and and discusses limitations of our approach. Section 5 provides an overview of related work. Finally, Section 6 concludes the paper.

2   A Typical Client-Side Attack

In a typical client-side web attack, the aim of the attacker is to take control of the user's web client in order to manipulate the client's interaction with the web application. Such an attack typically consists of three phases. In the first phase, the attacker's objective is to install malware on the user's computer. Once this has been successfully achieved, in the second phase, the installed malware monitors the user's interaction with the web application. The third phase starts once the malware detects that a security-critical operation is taking place and attempts to manipulate the flow of sensitive information to the web application to fulfill the attacker's objectives.

Imagine, for example, that John Smith receives an email with a link to a URL. This email has been sent by attackers to thousands of users. John is naive and curious, so he clicks on the link. Unfortunately, he has not regularly updated his browser (Internet Explorer in this case), which contains a serious parsing-related vulnerability that allows malicious code to be injected and executed on his system just by visiting a hostile web site. As a result, a Trojan horse is automatically installed on John's computer when his browser parses the contents of the web page.

The Trojan horse that the attackers have prepared is a Browser Helper Object (BHO) for the Internet Explorer (IE). This BHO is automatically loaded every time IE is started. With the BHO, the attackers have access to all events (i.e., interactions) and HTML components (i.e., DOM objects) within the browser. Hence, they can easily check which web sites the user is surfing, and they can also modify the contents of web pages. In our example, the attacker's are interested in web sessions with a particular bank (the Bank Austria).

Whenever John is online and starts using the Bank Austria online banking web application, the Trojan browser-plugin is triggered. It then starts analyzing the contents of the bank web pages. When it detects that he is about to transfer money to another account, it silently modifies the target account number.

Note that the imaginary attack we described previously is actually very similar to the attacks that have been recently targeting Austrian banks. Clearly, there can be many technical variations of such an attack. For example, instead of using a BHO, the attackers could also inject Dynamic Link Libraries (DLLs) into running applications or choose to intercept and manipulate Operating System (OS) calls.

The key observation here is that the online banking web application has no way to determine whether the client it is interacting with has been compromised. Furthermore, when the client has indeed been compromised, all security precautions the web application can take to create a secure communication channel to the client (e.g., TLS encryption) fail. That is, the web application cannot determine whether it is directly interacting with a user, or with a malicious application performing illegitimate actions on behalf of a user.

3   Our Solution

As described in the previous section, the web application must assume that the user's web client (and platform)

is under the control of an attacker. There are two aspects of the communication that an attacker could compromise: the confidentiality, or the integrity of input sent from the client to the web application. The confidentiality of the input is compromised when the attacker is able to eavesdrop on the entered input and intercept sensitive information. Analogously, the integrity of the input is compromised when the attacker is able to tamper, modify, or cancel the input the user has entered.

As far as the user is considered, there are cases in which the integrity of input may be more important than its confidentiality. For example, as described in Section 2, only when the attacker can effectively *modify* the account number that has been typed, an illegitimate money transaction causing financial damage can be performed.

In this section, we present two techniques that web applications can apply to protect sensitive user input. We assume a threat model in which the attacker has compromised a machine and installed malicious code. This code has complete control of the client's machine, but must perform its task in an autonomous fashion (i.e., without being able to consult a human). Our solution are implemented on the server and are client-independent. The first solution we discuss aims to protect the integrity of user input. The second solution we discuss aims to protect the confidentiality *and* integrity of the user input, but only against *automated attacks* (i.e., the adversary is not a human).

3.1 Solution 1: Binding Sensitive Information to Confirmation Tokens

3.1.1 Overview

The first solution is based on *confirmation tokens*. In principle, the concept of a confirmation token is similar to a transaction number (i.e., TANs) commonly used in online banking. TANs are randomly generated numbers that are sent to customers as hardcopy letters via regular (snail) mail. Each time a customer would like to confirm a transaction, she selects a TAN entry from her hardcopy list and enters it into the web application. Each TAN entry can be used only once. The idea is that an attacker cannot perform transactions just by knowing a customer's user login name and password. Obviously, TAN-based schemes rely on the assumption that an attacker will not have access to a user's TAN list and hence, be able to perform illegitimate financial transactions at a time of his choosing.

Unfortunately, TAN-based schemes are easily defeated when an attacker performs a client-side attack (e.g., using a Trojan horse as described in Section 2). Furthermore, such schemes are also vulnerable to phishing attempts in which victims are prompted to provide one (or more) TAN numbers on the phishing page. The increasing number of successful phishing attacks prompted some European banks to switch to so called *indexed TAN (i-TAN)* schemes, where the bank server requests a specific i-TAN for each transaction. While this partially mitigated the phishing threat, i-TANs are as vulnerable to client-side attacks as traditional TANs.

In general, the problem with regular transactions numbers is that there is no relationship between the data that is sent to the web application and the (a-priori shared) TANs. Thus, when the bank requests a certain TAN, malicious code can replace the user's input without invalidating this transaction number. To mitigate this weakness and to enforce integrity of the transmitted information, *we propose to bind the information that the user wants to send to our confirmation token*. In other words, we propose to use confirmation tokens that (partially) depend on the user data. Note that when using confirmation tokens, our focus is not the protection of the confidentiality, but the integrity of this sensitive information.

3.1.2 Details

Imagine that an application needs to protect the integrity of some input data x. In our solution, the idea is to specify a function $f(.)$ that the user is requested to apply to the sensitive input x. The user then submits both her input data x and, as a confirmation token, $f(x)$.

Suppose that in an online banking scenario, the bank receives the account number n together with a confirmation token t from the user. The bank will then apply $f(.)$ to n and verify that $f(n) = t$. If the value x, which the user desires to submit, is the same as the input n that the bank receives ($x = n$), then the computation of $f(n)$ by the bank will equal the computation of $f(x)$ by the user. That is, $f(x) = f(n)$ holds. If, however, the user input is modified, then the bank's computation will yield $f(n) \neq f(x)$, and the bank will know that the integrity of the user's input is compromised.

Any important question that needs to be answered is how $f(.)$ should be defined. Clearly, $f(.)$ has to be defined in a way so that malicious software installed on a user's machine cannot easily compute it. Otherwise, the malware could automatically compute $f(x)$ for any input x that it would like to send, and the proposed solution fails. Also, $f(.)$ has to remain secret from the attacker.

We propose two schemes for computing $f(x)$. For both schemes, the user will require a *code book*. This code book will be delivered via regular mail, similar to TAN letters described in the previous section. In the first scheme, called *token calculation*, the code book contains a collection of simple algorithms that can be used by users to *manually* compute confirmation tokens (similar to the obfuscation and challenge-response idea presented in [4] for secure logins). All algorithms are based on the input that the user would like to transmit.

```
...
Token ID 4:
    Create a number using the 5th and 7th
    digits of the target account and add 542 to it.
Token ID 5 :
    Create a number using the 3rd and 5th
    digits of the target account and add 262 to it.
Token ID 6:
    Multiply the 4th and 8th digits of the target
    account and add 17 to the result.
Token ID 7:
    Create a number using the 3rd, 6th and 7th
    digits of the target account.
...
```

Figure 1. Excerpt from our sample token calculation code book.

Suppose that the user has entered the account number 980.243.276, but a Trojan horse has actually sent the account number 276.173.862 to the bank (unnoticed by the user). In the first scheme, the bank would randomly choose an algorithm from the user's code book. Clearly, in order to make the scheme more resistant against attacks, a different code book would have to be created for each user (just like different TANs are generated for different users). Figure 1 shows an excerpt from our sample token calculation code book. Suppose the bank asks the user to apply algorithm ID 6 to the target account number. That is, the user would have to multiply the 4th and 8th digits of the account number and add 17 to the result. Hence, the user would type 31 as the confirmation token. The bank, however, would compute 23 and, because these confirmation values do not match, it would not execute the transaction, successfully thwarting the attack.
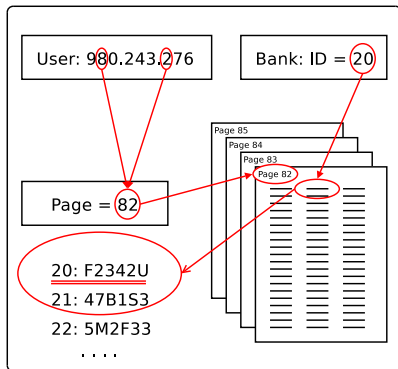
Figure 2. The token lookup scheme illustrated.

For our second scheme to implement $f(.)$, called *token lookup*, users are not required to perform any computation. In this variation, the code book would consist of a large

number of random tokens that are organized in *pages*. The bank and the user previously and secretly agree on *which* digits of the account number are relevant for choosing the correct page. The bank then requests the user to confirm a transaction by asking her to enter the value of a *specific* token on that page. For example, suppose that the relevant account digits are 2 and 7 for user John and that the bank asks John to enter the token with the ID 20. In this case, John would determine the *relevant* code page by combining the 2nd and 7th digits of the account number and look up the token on that page that has the ID 20 (Figure 2). Suppose that the user is faced with the same attack that we discussed previously. That is, the user enters 980.243.276, but the malicious application sends 276.173.862 to the bank. In this case, the user would look up the token with ID 20 on page 82, while the bank would consult page 78. Thus, the transmitted token would not be accepted as valid.

3.1.3 Security Analysis

The security of the confirmation-token-based solutions relies on the assumption that the function $f(.)$ remains secret. For the token computation, this implies that the algorithms the bank asks the user to apply are not revealed and cannot be deduced from the account number and the computed confirmation token. The lookup scheme can only be compromised when both the relevant digits for code lookups and the contents of the code book are disclosed.

Note that it is possible in theory that some digits of the target account of the attacker correspond to the digits in the target account number of the user. Hence, it could be possible that an algorithm is chosen by the bank that computes the *same* confirmation token for both account numbers, or that both account numbers produce the same page number for the lookup scheme. The probability for such an event is 1-in-10^n, with n being the number or relevant digits.

Another possibility to circumvent our schemes is trying to guess the right confirmation token. In our prototype implementation, the calculated token can be two to four digits long, giving a 1-in-11,100 chance of guessing the right one (note that `10` and `010` are two different tokens). For the lookup scheme, we used 6-digit alphanumeric tokens. The chance to guess the right one is 1-in-36^6.

3.2 Solution 2: Using CAPTCHAs for Secure Input

3.2.1 Overview

Graphical input is used by some banks and other institutions to prevent eavesdropping of passwords or PINs. Instead of using the keyboard to enter sensitive information, an image of a keypad is displayed, and the user enters data by clicking on the corresponding places in the image. Unfortunately,

these schemes are typically very simple. For example, the letters and numbers are always located at the same window coordinates, or the fonts can be easily recognized with optical character recognition (OCR). As a result, malware can still recover the entered information.

The basic idea of the second solution is to extend graphical input with CAPTCHAs [3]. A CAPTCHA, which stands for Completely Automated Public Turing test to tell Computers and Humans Apart, is a type of challenge-response test that is used in computing to determine whether or not the user is human. Hence, a CAPTCHA test needs to be solvable by humans, but not solvable (or very difficult to solve) for computer applications. CAPTCHAs are widely employed for protecting online services against automated (mis)use by malicious programs or scripts. For example, such programs may try to influence online polls, or register for free email services with the aim of sending spam. Figure 3 shows a graphical CAPTCHA generated by Yahoo when a user tries to subscribe to its free email service.

Figure 3. A graphical CAPTCHA generated by Yahoo.

An important characteristic of a CAPTCHA is that it has to be resistant to attacks. That is, it should not be possible for an algorithm to automatically solve the CAPTCHA. Graphical CAPTCHAs, specifically, need to be resistant to optical character recognition [18]. OCR is used to translate images of handwritten or typewritten text into machine-editable text. To defeat OCR, CAPTCHAs generally use background clutter (e.g., thin lines, colors, etc.), a large range of fonts, and image transformations. Such properties have been shown to make OCR analysis difficult [3].

Usually, the algorithm used to create a CAPTCHA is made public. The reason for this is that a good CAPTCHA needs to demonstrate that it can only be broken by advances in OCR (or general pattern recognition) technology and not by the discovery of a "secret" algorithm. Note that although some commonly used CAPTCHA algorithms have already been defeated (e.g., see [17]), a number of more sophisticated CAPTCHA algorithms [3, 7] are still considered resistant against OCR and are currently being widely used by companies such as Yahoo and Google.

3.2.2 Details

Although CAPTCHAs are frequently used to protect online services against automated access, to the best of our knowledge, no one has considered their use to enable secure input to web applications.

In our solution, whenever a web application requires to protect the integrity and confidentiality of user information, it generates a graphical input field with randomly placed CAPTCHA characters. When the user wants to transmit input, she simply uses the mouse to click on the area that corresponds to the first character that should be sent. Clicking on the image generates a web request that contains the *coordinates on the image* where the user has clicked with the mouse. The key idea here is that *only* the web application knows which character is located at these coordinates. After the first character is transmitted, the web application generates another image with a different placement of the characters, and the process is repeated. By using CAPTCHAs to communicate with the human user, a web application can mitigate client-side attacks that intercept or modify the sensitive information that users type. Because the CAPTCHA characters cannot be identified automatically, a malware program has no way to know which information was selected by the user, nor does it have a way to meaningful select characters of its own choosing.
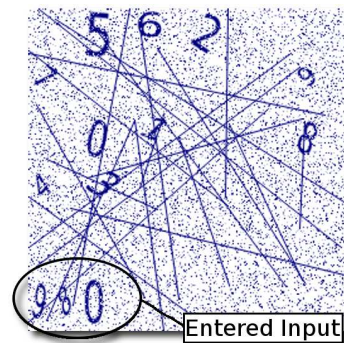
Figure 4. A generated CAPTCHA for enabling secure user input.

Figure 4 depicts a screenshot of our prototype CAPTCHA input element. In this example, the user would like to securely enter her bank account number 980.343.276. It can be seen that the web application has generated digits from 0 to 9 that are randomly distributed on the upper half of the image. To transmit the account information, the user locates the desired numbers on each generated image and sequentially enters the account number that she would like to send. In order to give immediate feedback to the user about the number she has just entered, a part of the CAPTCHA can be used to reflect back the input. In our prototype implementation, we chose to use the bottom half of the CAPTCHA to give feedback to the user about the number she has just entered (see the ellipse in Figure 4; the first three digits have already been entered). Of course, one can also allow

the user to modify the input she has entered by including input elements in the CAPTCHA that can be used for deleting and editing.

3.2.3 Security Analysis

The security of the CAPTCHA-based solution is based on the assumption that it is impossible (or sufficiently difficult) for malicious software installed on the user's machine to analyze and identify the coordinates on the CAPTCHA where the input values are located. Our solution also relies on the presumption that we can adapt and integrate recent advances in CAPTCHA research and hence, match advances in OCR research.

In order to successfully defeat our solution automatically, the malicious software would need to identify the mouse coordinates that it needs to send to the web application. Suppose, for example, that a Trojan horse would like to send the account number 276.273.862 to the bank instead of the original account number 980.343.276 that the user would like to enter. For each new CAPTCHA input element, it would have to automatically identify the target input value that it requires, intercept the user's mouse click, and simulate a mouse click of its own. In our example, the Trojan would first have to identify the number 2 in the first generated CAPTCHA, the number 7 on the second generated CAPTCHA, and so on.

Although algorithms exist that can make it sufficiently difficult for software to solve CAPTCHAs automatically, the main problem currently is that it may be possible to relay CAPTCHAs to human operators who can then solve them manually. In fact, some spammers have been using this technique in real-life to defeat CAPTCHAs [26]. One popular idea is to divert the CAPTCHA to another web site where visitors willingly solve them because they believe that they will access "free" adult content. Note, however, that in the solution we propose, it is more difficult to deploy an unsuspecting human operator to defeat the CAPTCHA. That is, in our case, the required task is not simply reading the CAPTCHA, but identifying input values sequentially. Hence, a human operator would have to follow the input that is being entered in real-time while the victim is entering this information during an online banking session.

The attackers could nevertheless deploy human operators that indeed analyze and enter information on CAPTCHA-based input elements. One can imagine, for example, that the miscreants would employ and pay these operators for their services. When human operators are involved, the confidentiality of the input can be compromised, since nothing will prevent an off-line analysis of the images and click coordinates. However, an effective attack on the integrity can only happen on-line (i.e., at the moment the user attempts to enter the sensitive data). In this case, the

attackers would have to make sure that the communication between the Trojan horses on the victims' machines and their human operators are reliable and fast. Furthermore, they would also run the risk of being traced back if service providers monitor network connections and detected Trojan horses are analyzed for their behavior. Finally, to effectively deceive the user, that part of the image that reflects the input needs to be edited (to show the input the user *intended* to make) before being displayed. Thus, we believe that the CAPTCHA-based input solution we propose considerably raises the difficulty bar for attackers and significantly increases the costs of client-side attacks.

4 Evaluation

In Section 3, we discussed a number of possibilities an attacker possesses to subvert our protection schemes. The discussion showed that our system offers appealing security properties even in the presence of powerful adversaries. However, to be usable in practice, it is not sufficient that a scheme is secure, but it is also necessary that it is easy to understand and to use. To assess the ease-of-use of our proposed protection mechanisms, we conducted two user studies. We performed an initial small-scale study to identify major usability obstacles of our prototype implementation. Based on these initial experiences, we improved our system and eventually conducted a larger user study to evaluate how suitable our protection mechanisms are to protect online bank transactions.

To conduct the usability study, we implemented a simple proof-of-concept online banking web application, where a user can log in and perform (simulated) money transactions. Our transaction functionality is similar to real-life online banking web applications where the user is required to enter a target account number, routing number and the amount that is to be transferred. After entering the transaction details, the confirmation challenge is displayed. If the user enters an invalid token, she has the chance to retry indefinitely. At any point, the user has the chance to abort.

For the initial study, we recruited 16 participants with diverse background and age. Each participant was asked to log into our prototype bank site and carry out a number of transactions. To this end, every user received a list of ten transactions that she should perform. For each transaction, the user was told to select one of the three available protection schemes, that is, the users had the choice between either CAPTCHA input, token lookup, or token calculation.

On the login page, a help text (about 2 screen pages long) was displayed that explained the usage of the new protection methods. Initially, no additional information was given. When a user failed to correctly perform a transaction and gave up, the experimenter took on the role of a customer support agent and was allowed to answer some

specific questions or to address a particular problem that the subject encountered. For each protection technique, a number of transactions were performed. We silently observed the problems that subjects encountered before any hints were given, and also after some support was provided. At the end, we asked each person to rank the protection mechanisms with regard to their personal feeling of ease-of-use. Also, we were interested in suggestions that people might have to improve their user experience.

The initial study yielded the following findings: The token lookup scheme was considered easiest, while token calculation was the least favorite method for almost all users. Problems with the latter resulted from difficulties of understanding the textual descriptions of the algorithms, but also from frustrating miscalculations.

For the graphical input method, there was a negative correlation between popularity and age, while participants with a technical background generally had a better appreciation for it. We concluded that the CAPTCHA input was "over-featured." The problem was a single, large input image that contained several lines of input lumped together (one line for each information needed for a transaction), as well as navigation and editing elements. That proved to be confusing for most users. Additionally, some participants reported that the digits "1" and "7" were often hard to distinguish.

In general, we observed that, initially, most users had difficulties to understand how our protection mechanisms work. Nevertheless, after users overcame the initial obstacles and developed some familiarity with a new technique, they were typically able to carry out transactions swiftly and with few problems. It became apparent that, instead of a long introductory text, an (interactive) tutorial would be preferable to introduce each method. Finally, we realized that it makes little sense to compare the graphical input directly with the token schemes; both approaches address the same problem, but from a different angle.

For the second study, we redesigned out testing framework to mitigate the previously identified issues. We replaced the written introduction with a tutorial where the user was guided step-by-step through the first transaction with every scheme. To improve token calculation, we attempted to find more precise wording for the algorithm descriptions, but we had to balance the verbosity and the length of the instructions. Since we have concluded that graphical input and confirmation tokens are independent, we have implemented the methods to be used orthogonally. The user now has the choice between three *confirmation* methods: The two novel token schemes presented in Section 3.1 as well as the regular TAN method. Additionally, the user can activate the CAPTCHA graphical input (Section 3.2) for entering sensitive information. This allowed us to compare graphical input with normal keyboard input. Note that token schemes and the CAPTCHA-based input can be combined, for ex-

ample, by entering the account number via a CAPTCHA and using a lookup token to confirm the transaction.

To address the issues specific to graphical input, we have changed the user interface substantially. In particular, we replaced the original input image with a set of images, each displaying a single piece of information (such as account number or amount). Clicking on such an image reveals the graphical input panel. The panel contains only the digits, while the editing elements are realized as buttons. The user input is still reflected at the bottom of the input image, but the displayed digits are also valid input elements now (digits in the image that did not "respond" to clicks were often a source of confusion).

For the second study, we were able to recruit 64 participants. Again, we tried to accommodate diverse backgrounds and age groups. We included only people with (at least) basic computer skills and good knowledge of the German language, to prevent computer and language-related problems from influencing the outcome. The participants were between 18 and 50 years of age, 59% had some sort of technical education, and 43% declared their computer skills to be above average. Finally, 84% had previous online banking experience.

At the beginning, each participant was issued a task list, containing the login data and a list of twelve transactions, as well as the code books/TAN lists required for the confirmation schemes. The experimenter explained the task briefly, without giving any specifics, but pointing to the tutorial instead. Then, the users were left alone and we observed their actions unobtrusively. The tutorial consists of three transactions. The first introduces the CAPTCHA input and TAN confirmation, while the other two present the token lookup and token computation schemes, respectively. If, during the tutorial, a user asked for assistance, the experimenter would take the role of a customer support agent. In addition to the three introductory transactions, the task list contained nine additional transactions, which had to be carried out using the specified input and confirmation methods (in different combinations). From the overall twelve transactions, six were carried out using graphical input. Of the three confirmation schemes (TAN, token lookup, and token calculation), each had to be performed four times.

After completing all transactions, the users were pointed to an on-line questionnaire, with questions concerning the general difficulty of each method, as well as method-specific questions. In particular, we asked to rate the ease-of-use of each method (independently), providing four grades ranging from "easy" to "very hard." Moreover, we wanted to know if a task became easier with time, with the choices being "yes," "partially," and "no." The answers are listed in Table 1. In this table, the "Total" columns list the absolute difficulties of each scheme. The adjacent columns break down the relative improvement for each group. For

| Method | Graphical Input | | | | Token Lookup | | | | Token Calculation | | | |
Improvement Difficulty	Total	Yes	Part.	No	Total	Yes	Part.	No	Total	Yes	Part.	No
Easy	21	15	2	4	44	28	4	12	12	7	1	4
Medium	39	30	5	4	20	15	3	2	39	18	14	7
Hard	3	1	2	0	0	0	0	0	13	3	3	7
Very hard	1	0	1	0	0	0	0	0	0	0	0	0
Total	64	46	10	8	64	43	7	14	64	28	18	18

Table 1. Breakdown of the difficulties and learning curves reported by the study participants.

example, of the 39 users that considered the graphical input to be medium difficult, 30 found that this input method became easier after getting used to it. It can be seen that only few consider any of the methods to be hard, with token calculation being rated least favorably. An important finding is that of those who did not consider the methods to be easy initially, a majority reported that the experience improved over time.

To see if the users' responses to the questionnaire coincide with actual behavior, we compared the questionnaire results with the logs recorded during our study. Indeed, for the graphical input method, we observed that the input duration and input errors (i.e., the user missed the digit or entered a wrong digit) decreased with time. The average time to enter a 9-digit account number decreases from about 56 seconds to about 23 seconds. The average click interval (i.e., the time it takes to find the next desired digit in the image) goes down from the initial 3.8 seconds to 2.1 seconds and can be as low as 1.5 seconds for certain numbers. For the confirmation schemes, we recorded for each user and transaction the number of token resubmissions until the transaction was completed successfully. Also in this case, we observed that the number of necessary retries decreased over time. For the token lookup method, the average number of retries decreased from 0.21 for the first attempt to 0.05 after three rounds. The decrease for the token calculation scheme was from initially 0.68 retries down to 0.24.

Method	Yes	Sometimes	No
Graphical Input	17	24	23
Token Lookup	4	18	42
Token Calculation	15	25	24

Table 2. The "annoyance factor" — did using the particular method take too long?

To assess the "annoyance factor," we asked, for each method, whether performing a transaction would take "too long." The results are shown in Table 2. We can see that the majority does not consider the extra effort to be overly annoying. Finally, we asked whether the study participants would be willing to use the proposed methods for their actual online banking, provided that these methods offer additional security. For CAPTCHA input, 51 participants (80%) answered "yes," while for the confirmation schemes, the approval was even higher (58 users or 91%).

When comparing the results from this study with our initial one, we can conclude that the additional effort invested in improving our prototype has paid off. While in the first study, assistance was requested by almost all participants, now only ten people required assistance with the CAPTCHA input, seven needed help with token computation, and two with token lookup. Moreover, during the initial study, 87% of the transactions were completed successfully, while in the new study, the success rate was raised to 95%. Finally, we compared the average durations for a (successful) transaction: 133 seconds in the first study compared to 94 seconds in the second. When we exclude the tutorial transactions from the average, the time even drops to 77 seconds.

In addition to the ease-of-use, our study was also designed to assess the security of schemes that require some collaboration (and alertness) from users. To this end, we simulated a Trojan horse that attempted to tamper with some of the transactions (i.e., change the destination account number). We designed the Trojan to do this openly, that is, the modified account number was displayed on the confirmation page (and when entering it via CAPTCHA input). The hope (from the point of view of the attacker) was that the user would not notice the malicious account number and complete the transaction anyways. We did not specifically advise users of the fact that the machine could be compromised. For each user, the Trojan allowed the first few transaction to be free of tampering. Then, attacks were injected randomly with a certain probability. In the first study, the probability was high (causing up to three attacks for each user), and we decided to lower it for the second study because we realized that too many attacks distracted the users from the main goal of this study, assessing the system's usability. Now, about 25% of the participants experienced a single attack in the course of the test run.

We did not assume that this unsophisticated attack would

prove effective, but 25% of the attacks in the first study were successful, regardless of the security scheme used. In the second study, we noted that all attacks against CAPTCHA input were noticed by the users. Without graphical input, a few attacks were successful. This underlines in a disturbing fashion the limits of security solutions that require user cooperation. Nevertheless, our proposed protection techniques force an attacker to trick a user to accept a clearly incorrect account number. This is in contrast to current solutions, where the attack is invisible.

Given the results of our evaluations and our experiences while conducting the user studies, it was apparent that once our proposed solution was understood, most people were able to perform the required steps with little difficulty. Considering the additional protection that our techniques provide, we believe that they are suitable for deployment in security-critical environments such as online banking.

5 Related Work

Client-side sensitive information theft (e.g., spyware, keyloggers, Trojan horses, etc.) is a growing problem. In fact, the Anti-Phishing Working Group has reported over 170 different types of keyloggers distributed on thousands of web sites [1]. Hence, the problem has been increasingly gaining attention and a number of mitigation ideas have been presented to date.

Several client-side solutions have been proposed that aim to mitigate spoofed web-site-based phishing attacks. Pwd-Hash [22] is an Internet Explorer plug-in that transparently converts a user's password into a domain-specific password. A side-effect of the tool is some protection from phishing attacks. Because the generated password is domain-specific, the password that is phished is not useful. SpoofGuard [5] is a plug-in solution specifically developed to mitigate phishing attacks. The plug-in looks for "phishing symptoms" such as similar sounding domain names and masked links. Note that both solutions focus on the mitigation of spoofed web-site-based phishing attacks. That is, they are vulnerable against client-side attacks as they rely on the integrity of the environment they are running in. Similarly, solutions such as the recently introduced Internet Explorer anti-phishing features [16] are ineffective when an attacker has control over the user's environment.

Spyblock [11] aims to protect user passwords against network sniffing and dictionary attacks. It proposes to use a combination of password-authenticated key exchange and SSL. Furthermore, as additional defense against pharming, cookie sniffing, and session hijacking, it proposes a form of transaction confirmation over an authenticated channel. The tool is distributed as a client-side system that consists of a browser extension and an authentication agent that runs in a virtual machine environment that is "protected" from spy-

ware. A disadvantage of Spyblock is that the user needs to install and configure it, as opposed to our purely server-side solution.

A number of hardware-based solutions have been proposed to enable secure input on untrusted platforms. Chip cards and smart-card readers [10, 23], for example, are popular choices. Unfortunately, it might be possible for the attacker to circumvent such solutions if the implementations rely on untrusted components such as drivers and operating system calls [12, 24, 25]. As an alternative to smart-card-based solutions, several researchers have proposed using handhelds as a secure input medium [2, 15]. Note that although hardware-based solutions are useful, unfortunately, they are often expensive and have the disadvantage that they have to be installed and available to users.

A popular anti-keylogger technique that is already being deployed by certain security-aware organizations are graphical keyboards. Similar to our graphical input technique, the idea is that the user types in sensitive data using a graphical keyboard. As a result, she is safe from keyloggers that record the keys that are pressed. However, there have been increasing reports of so-called "screenscrapers" that capture the user's screen and send the screenshot to a remote phishing server for later analysis [6]. Also, with many graphical keyboard solutions, sensitive information can be extracted from user elements that show the entered data to provide feedback for the user. Finally, to the best of our knowledge, no graphical keyboard solution uses CAPTCHAs. Thus, the entered information can be determined in a straightforward fashion using simple OCR schemes.

The cryptographic community has also explored different protocols to identify humans over insecure channels [8, 14, 27]. In one of the earliest papers [14], a scheme is presented in which users have to respond to a challenge, having memorized a secret of the modest amount of ten characters and five digits. The authors present a security analysis, but no usability study is provided (actually, the authors defer the implementation of their techniques to future work). The importance of usability studies is shown in a later paper by Hopper and Blum [8]. In their work, the authors develop a secure scheme for human identification, but after performing user studies with 54 persons, conclude that their approach "is impractical for use by humans." In fact, a transaction takes on average 160 seconds, and can only be performed by 10% of the population. Our scheme, on the other hand, takes less than half of this time, and 95% of the transactions completed successfully.

Finally, client-side attacks could be mitigated if the user could easily verify the integrity of the software running on her platform. Trusted Computing (TC) [20] initiatives aim to achieve this objective by means of software and hardware. At this time, however, TC solutions largely remain prototypes that are not widely deployed in practice.

6 Conclusion

Web applications have become the most dominant way to provide access to online services. A growing class of problems are client-side attacks in which malicious software is automatically installed on the user's machine. This software can then easily access, control, and manipulate all sensitive information in the user's environment. Hence, an important web security research problem is how to enable a user on an untrusted platform to securely transmit information to with a web application.

Previous solutions to this problem are mostly hardware-based and require peripheral devices such as smart-card readers and mobile phones. In this paper, we present two novel server-side techniques that can be used to enable secure user input. The first technique uses confirmation tokens that are bound to sensitive data to ensure data integrity. Confirmation tokens can either be looked up directly in a code book or they need to be calculated using simple algorithms. The second technique extends graphical input with CAPTCHAs to protect the confidentiality and integrity of user input against automated attacks. The usability studies that we conducted demonstrate that, after an initial learning step, our techniques are understood and can also be applied by a non-technical audience.

Our dependency on the web will certainly increase in the future. At the same time, client-side attacks against web applications will most likely be continuing problems as the attacks are easy to perform and profitable. We hope that the techniques we present in this paper will be useful in mitigating such attacks.

Acknowledgments

References

[1] Anti-phishing Working Group. http://www.antiphishing.org.

[2] D. Balfanz and E. Felten. Hand-Held Computers Can Be Better Smart Cards. In *Proceedings of the 8th Usenix Security Symposium*, 1999.

[3] Carnegie Mellon University. The CAPTCHA Project. http://www.captcha.net.

[4] W. Cheswick. Johnny Can Obfuscate: Beyond Mother's Maiden Name. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Security (HotSec)*, 2006.

[5] N. Chou, R. Ledesma, Y. Teraguchi, and J. C. Mitchell. Client-side defense against web-based identity theft. In *Proceedings of the Network and Distributed Systems Security (NDSS)*, 2004.

[6] FinExtra.com. Phishers move to counteract bank security programmes. http://www.finextra.com/fullstory.asp?id=14149.

[7] S. Hocevar. PWNtcha - Captcha Decoder. http://sam.zoy.org/pwntcha.

[8] N. Hopper and M. Blum. Secure Human Identification Protocols. In *AsiaCrypt*, 2001.

[9] IETF Working Group. Transport Layer Security (TLS). http://www.ietf.org/html.charters/tls-charter.html, 2006.

[10] International Organization for Standardization (ISO). ISO 7816 Smart Card Standard. http://www.iso.org/.

[11] C. Jackson, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Virtual Machines. http://crypto.stanford.edu/SpyBlock/spyblock.pdf.

[12] A. Josang, D. Povey, and A. Ho. What You See is Not Always What You Sign. In *Annual Technical Conference of the Australian UNIX and Open Systems User Group*, 2002.

[13] I. Krawarik and M. Kwauka. Attacken aufs Konto (in German). http://www.ispa.at/www/getFile.php?id=846, Jan 2007.

[14] T. Matsumoto and H. Imai. Human Identification Through Insecure Channel. In *EuroCrypt*, 1991.

[15] J. M. McCune, A. Perrig, and M. K. Reiter. Bump in the Ether: A Framework for Securing Sensitive User Input. In *Proceedings of the USENIX Annual Technical Conference*, June 2006.

[16] Microsoft Corporation. Internet Explorer 7 features. http://www.microsoft.com/windows/ie/ie7/about/features/default.mspx.

[17] G. Mori and J. Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. In *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference (CVPR)*. IEEE Computer Society Press, 2003.

[18] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Document image analysis*, pages 244–273, 1995.

[19] A. Moshchuk, T. Bragin, S. D. Gribble, and H. M. Levy. A Crawler-based Study of Spyware on the Web. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS)*, February 2006.

[20] S. Pearson. *Trusted Computing Platforms*. Prentice Hall, 2002.

[21] Pressetext Austria. Phishing-Schäden bleiben am Kunden hängen (in German). http://www.pressetext.at/pte.mc?pte=061116033, Nov 2006.

[22] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger Password Authentication Using Browser Extensions. In *Proceedings of the 14th Usenix Security Symposium*, 2005.

[23] Secure Information Technology Center Austria (A-SIT). The Austrian Citizen Card. http://www.buergerkarte.at/index_en.html, 2005.

[24] A. Spalka, A. Cremers, and H. Langweg. Protecting the Creation of Digital Signatures with Trusted Computing Platform Technology Against Attacks by Trojan Horse. In *IFIP Security Conference*, 2001.

[25] A. Spalka, A. Cremers, and H. Langweg. Trojan Horse Attacks on Software for Electronic Signatures. *Informatica*, 26, 2002.

[26] W3C Working Group. Inaccessibility of CAPTCHA, Alternatives to Visual Turing Tests on the Web. `http://www.w3.org/TR/turingtest/`.

[27] C. Wang, H. Hwang, and T. Tsai. One the Matsumoto and Imai's human identification scheme. In *EuroCrypt*, 1995.