Usability Testing a Malware-Resistant Input Mechanism

Alana Libonati
University of North Carolina
alana@cs.unc.edu

Jonathan M. McCune
Carnegie Mellon University
jonmccune@cmu.edu

Michael K. Reiter
University of North Carolina
reiter@cs.unc.edu

Abstract

We report the results of a usability study of Bumpy, a system that enables a user to provide secret inputs to remote webservers without trusting the computer on which she types those inputs. Achieving this somewhat paradoxical property via Bumpy requires extra diligence from users, raising questions as to whether it is a viable protection for the average user. We evaluate the originally proposed Bumpy design and several new alternatives in a user study involving 85 participants, each of whom utilized one of these designs (or a control design) for roughly four months to protect her password entries to a university course web page. Beyond assessing the usability of Bumpy designs, our study offers insights for designing security-relevant interfaces and training users to successfully utilize them.

1 Introduction

Malware infections are commonplace (e.g., [5, 14, 20]), suggesting that a considerable percentage of users will sit down to an infected computer at some point. There are few defenses that help a user cope with such situations, particularly if the malware has infiltrated the lowest levels of the operating system (or virtual machine monitor [15, 24, 34], or lower [7, 35]). Those defenses that do must somehow instill in each user a degree of mistrust in the computer she is using, so that the user will employ a level of suspicion that allows her to detect malware behaviors and / or avoid divulging any further sensitive information to the malware.

One defense intended for such a circumstance is Bumpy [17], a mechanism that enables a user to convey secret information to remote webservers through a host without trusting that host with the information itself. Briefly, Bumpy works by first processing user keystrokes in a small, trusted module that is isolated from all other software on the computer using newly available hardware-enforced protections [12]. Because this module receives user inputs before the other software does, the user can indicate an input that she considers sensitive by preceding it with a *secure attention sequence* (SAS), causing the Bumpy module to treat it differently from normal inputs (which it passes on to the untrusted platform). In particular, Bumpy can queue sensitive inputs and prepare them (e.g., encrypt them) for a target destination, releasing only nondescript characters (e.g., asterisks) to the untrusted platform in their place. Aside from the user remembering to type the SAS to indicate the imminent arrival of a sensitive input, a central challenge in the Bumpy design is enabling the user to verify for what destination site the input will be prepared and sent. Since the untrusted platform itself, including the display, is presumed to be controlled by malware, the original Bumpy design [17] suggested conveying this information to the user via a separate device, termed a *Trusted Monitor* (TM). For example, the TM could be implemented using a dedicated token in the form factor of a flash drive, or it could even be implemented by a user's cell phone (albeit with the risks that would come with employing a general-purpose device potentially with its own vulnerabilities).

Typing a SAS and checking a TM for the proper destination for the forthcoming sensitive input might seem straightforward, particularly to computer security researchers. However, faithfully following this procedure and reacting appropriately in the face of unexpected results — potentially caused by the malware that Bumpy is trying to protect against — requires that the user adopt a degree of suspicion of her own computer that, to our knowledge, has not previously been achieved among non-expert users. As such, if malware can convince users to disclose their sensitive data despite the presence of Bumpy (i.e., by causing them to misuse the system), then there is little point in deploying the system.

In this paper, we evaluate the usability of the original Bumpy design, hereafter referred to as Original, alongside three other novel designs that we propose here. In the first, called Graphical, the entry of the SAS is replaced by a graphical process for indicating forthcoming sensitive input. In the second, called NoTM, the user enters a *per-site* SAS to proactively instruct Bumpy as to which site the subsequent sensitive input should be sent, thereby rendering the TM unnecessary. In the third, called Challenge, the SAS is

entered as in Original, after which the TM displays a challenge that the Bumpy module expects to receive prior to the sensitive input, thereby making inspection of the TM a non-bypassable part of entering sensitive input.

We evaluate each of the four designs both in terms of its imposition on the normal login process (impact on delay and success rate) and in terms of users' abilities to benefit from its protections despite the contrary efforts of malware. We performed this evaluation with a four-month user study involving 85 participants, the overwhelming majority of which were undergraduate college students majoring in fields other than computer science or engineering. Each of these 85 students was assigned to use one of the four Bumpy variants described above to log into a course web page for an entire semester (i.e., to protect her course login password) or to use a regular password login (Control). In this user study, we subjected users to phases of normal system use and to phases in which we simulated potent malware attacks to trick them into divulging their passwords. We followed the attack phase with an additional phase where users were warned about the mistakes they made when faced with simulated attacks. These warnings served as a form of training, and we analyze their effectiveness through additional simulated attacks.

Through these phases, we were able to identify relative strengths and weaknesses of our designs and quantify the degree of protection for sensitive data that the various Bumpy designs can offer. Notably, we uncovered evidence that the new designs that we introduce here yield the best results for many of the measures we studied, thereby improving over the original Bumpy design. That said, our study does not conclude that one design is uniformly better than all others. For example, the evidence suggests that some designs are more secure while others yield faster login times.

Viewed more broadly, our study provides a number of insights that may be useful in other contexts. Our study provides support for the usability of secure attention sequences and to the better security afforded by interactive security indicators (versus ones that a user is asked to simply observe). It also demonstrates that passive warnings are only partially effective in molding user behavior. We also believe our study supports the conclusion that without training, users will face difficulties in distinguishing changes in security-sensitive procedures (in our case, induced by malware attacks) from common discontinuities in the software experience, i.e., failures or updates that change software behavior, often in subtle ways.

2 Related Work

Bumpy falls within the general class of approaches to address the challenge of securely interacting with or through an untrusted computer (e.g., [1, 2, 4, 11, 13, 18, 19, 23, 26, 27]). We believe that the results of our study can inform other usability studies in this domain and potentially the design of alternative user experiences in this domain. In particular, common to many of these schemes is a secondary, trusted device, such as the user's smartphone or a dedicated USB device. In Bumpy's original design (Original), this secondary trusted device is the Trusted Monitor. The alternative user experiences that we develop here — one that eliminates the use of a TM (NoTM) and two others that retain the TM but that alter the user experience (Graphical and Challenge) — have parallels in other schemes, and our usability results should be instructive in selecting from among candidate designs elsewhere. Our results also underscore the need to perform usability studies of these alternative architectures to better understand the relative usability advantages of architectures based on entering sensitive input into a trusted mobile device (e.g., [1, 4, 11, 18, 19, 13, 26, 27]), leveraging a trusted remote proxy in the "cloud" (e.g., [23]), and employing a trusted virtualization layer (e.g., [2, 11]).

Bumpy's user experience bears some similarity to techniques developed to generate distinct domain-specific passwords from a single password, either for protecting privacy (e.g., LPWA [10]) or to defend against phishing attacks (e.g., PwdHash [22]). Bumpy's similarity to these systems lies primarily in the user's input of a secure attention sequence to convey instructions to the trusted module that produces the domain-specific passwords. Chiasson et al. performed a usability study of PwdHash and concluded that it provides insufficient user feedback [3]. Users were unable to discern whether PwdHash was actually working, resulting in situations where users reported that PwdHash was easy to use and protecting their passwords when in fact protections were disabled. This finding helps to motivate the presence of a feedback mechanism in three of the four Bumpy designs we considered.

Considerable work has been done regarding security indicators and how they are often unnoticed or misinterpreted by users (e.g., [6]). Researchers have tracked users' eye movements [29, 33] to determine whether they look at passive security indicators [9, 36], often finding that even the complete disappearance of common security indicators is ignored [25]. As the user is asked to pay attention to the passive website destination indicator on her TM to thwart attacks when using Original, these lessons from previous studies motivate the other designs that we explore here. One such design (Challenge) seemingly *requires* the user to pay attention to the indicator so that she can transcribe two digits, and another (NoTM) eliminates the indicator and enables the user to gain assurance proactively. Our study can inform others as to the effectiveness of such strategies to combat user ambivalence towards security indicators.

Interactive training methods have had success in educating users about security attacks (e.g., [28]). This motivated

our evaluation of a similar training regimen in one phase of our study.

3 The Bumpy User Experience

Here we describe the user experience of the original Bumpy design, denoted Original, and how our other designs Graphical, NoTM, and Challenge might appear to the user as implemented with the low-level security mechanisms leveraged by the actual Bumpy system. Here we focus on the user experiences exclusively, deferring to §4.1 the details of how we simulated these user experiences in our study.

3.1 Original Design – Original

Bumpy is motivated by the desire to allow users to protect arbitrary input to web forms in the face of malware on the user's local system, e.g., keyloggers. Arbitrary input means that the user can actively choose which data to protect. Logical choices include, but are not limited to, passwords and financial account numbers.

Users are equipped with an external device called a Trusted Monitor (TM). In practice this may be a dedicated device, or it may be functionality offered by, e.g., smartphones. The TM serves as a trusted output device for information about the user's input, as the Bumpy design allows for the user's OS to be malicious. The TM receives this input (in a cryptographically protected way) from a *Bumpy module* that processes user inputs before releasing them to the untrusted platform. The Bumpy module itself is protected from the OS with hardware-enforced isolation (see Appendix A).

Consider a user who is providing input to a web form. For example, she may be trying to log into an access-controlled website, or she may be going through the checkout process while making an online purchase. While interacting with a web form, the Bumpy user takes the following steps:

The user decides to protect the current form field. Allowing users to choose which data to protect is a feature of Bumpy's design. The following steps assume the user has chosen to protect the current field.

The user prefixes her input with a secure attention sequence (SAS). The Bumpy module detects that the user is preparing to provide data that she considers sensitive when it receives a SAS. Bumpy uses `@@` as the SAS. Thus, the user's responsibility is to prefix her forthcoming sensitive input with `@@`.

The user verifies the destination for her input on the Trusted Monitor. When the Bumpy module receives a SAS, it updates the TM with information about the destination for forthcoming input. We chose to use the domain name of the destination website and that website's favicon as a graphic logo for the website, since this information is already readily available.[1] The TM is further specified to beep when updated, thereby providing two properties: (i) It attracts the user's attention and reminds her to check the information displayed on the TM; and (ii) It provides a timely acknowledgment of the reception of the user's SAS. Thus, the user's responsibility is to verify that the site information displayed by the TM does indeed correspond to the website where she wants her input to go. While the user's primary display may be controlled by malware and display anything an attacker desires, the TM will always display where sensitive input will truly go.

The user types her input. If satisfied with the information displayed on the TM, the user now types her secret input. This input will always appear as asterisks, even if the current input field is not a password field, as the keystrokes are intercepted and queued by the Bumpy module, with asterisks released to the untrusted platform in their place. She may use the backspace or arrow keys to correct mistakes. The end of sensitive input is signaled to the Bumpy module by an input event that would cause a *blur* in the web browser's GUI. As such, the user does not need to consciously indicate that she has finished sensitive input to a given field. However, because Bumpy interprets input events that cause blurs to be meaningful, switching between input fields before they are fully populated interrupts the secure input process.

In summary, the main changes to the user experience are: (a) She must decide which data to protect using Bumpy. (b) She must remember to enter the SAS and confirm that the TM displays her intended destination for her input, before entering that data. (c) Her sensitive input appears on-screen as asterisks.

3.2 Alternative Designs

We now discuss the user experiences of our novel alternative designs Graphical, NoTM, and Challenge, as informed by the underlying hardware-enforced isolation technologies employed by the original Bumpy architecture. These designs differ from Original primarily in how a user signals to Bumpy that forthcoming input should be treated as sensitive. In all cases, the end of sensitive input is signaled to the Bumpy module by an input event that would cause a blur in the web browser's GUI.

[1] An alternative design is to allow this information to be chosen by the user, perhaps in the spirit of PetNames [30]. We caution that if the information can be freely specified by the destination website, then an attacker may register a legitimate SSL certificate for a site under his control and imitate the information of another site. Bumpy proposed drawing this information from the website's SSL certificate – see Appendix A for details.

3.2.1 Graphical SAS – Graphical

This design alternative tests whether a graphical means to denote sensitive input is more usable than a character-based SAS (@@). Instead of prefixing sensitive input with a SAS, users double-click within an input field to toggle its sensitivity. The sensitivity of an input field is indicated by its background color. We use green to indicate a field that is denoted sensitive, and red to denote a field that is unprotected (the default). This design retains the TM and the operation of the TM is unchanged from Original. The TM updates its display when the user's double-click within a field toggles that field to sensitive.

We caution that this design alternative is challenging to implement with the original Bumpy architecture because of its dependence on a trustworthy GUI — a requirement that is at odds with the assumption of malware on the user's computer. However, we believe it is important to discover whether a graphical SAS is significantly more usable than the character-based SAS of Original, NoTM, and Challenge, and so we included this test despite its implementation challenges.

3.2.2 No Trusted Monitor – NoTM

This design alternative eliminates the need for a TM. Without a TM, there is no trustworthy path through which the user can receive feedback about the destination for her forthcoming sensitive input. NoTM addresses this challenge by supporting a unique SAS for each destination, thereby allowing the user to specify the destination for her forthcoming sensitive input proactively. Instead of @@, the SAS is defined to be @*str*@, where *str* is a user-chosen string. The user can assign a particular string to a given website by using the $ character instead of the @, i.e., $*str*$ will assign @*str*@ to be the SAS for the active website. Note that this is a trust-on-first-use model, in that there is no feedback mechanism by which the user can gain additional assurance of which site is bound to @*str*@. While problematic at a completely unfamiliar computer like an internet kiosk, this model is useful for computers that the user employs regularly but still does not completely trust (such as a shared family computer).

While at first glance this need to remember a per-site SAS may seem to place a burden on the user's already overtaxed memory, *str* does not need to be secret. Only its integrity is required, and this is readily achieved because the Bumpy module processes the user's input stream before the untrusted platform. Thus, a reasonable convention is for *str* to be the first few characters of the destination's domain name, e.g., ama for amazon.com. The user would then enter ama upon her first visit to amazon.com, and subsequently prefix sensitive input intended for amazon.com with @ama@. Note that by first visit, we truly mean one visit; Bumpy will remember the SAS assignments between sessions. A user can even have more than one SAS for the same website.

The intention of this design is to determine whether allowing users to specify destinations for their sensitive inputs proactively is a more usable interface for Bumpy. There is reason to predict that NoTM would be comparatively effective, given the propensity of users to ignore passive security indicators (e.g., SSL indicators [31] and anti-phishing toolbars [36]) and given that prefixing a sensitive input with a per-destination SAS can be viewed as a type of direct navigation (i.e., keyboard entry of a familiar website URL).

3.2.3 Random Challenge – Challenge

This design alternative is intended to *require* the user to look at the TM. This is again motivated by the tendency of users to ignore passive security indicators. Through this design we seek to determine whether being required to look at the TM will increase the likelihood that users will notice if the TM is displaying an incorrect destination domain and graphic logo.

To force the user to examine the TM, Challenge displays a random 2-digit challenge value xy on the TM that must be entered as part of the SAS for this login attempt. More specifically, after the user clicks into a particular input field and enters @@ as the first part of the SAS, the TM will update its display to include a random two-digit challenge xy sent to it by the Bumpy module. The user then enters this challenge as the remainder of her SAS, yielding the full SAS @@xy, and then immediately follows this with her sensitive input. If it does not receive the values xy immediately following the @@, the Bumpy module will discard the sensitive input characters, thereby ensuring that they are not leaked to malware.

4 User Study Methodology

We now describe the methodology that we used to conduct our user study. Our study was focused on the usability of each Bumpy variant in benign scenarios, the security of each variant in attack scenarios, and the utility of warnings for training users to handle attack scenarios. We describe our simulation of each variant in §4.1 and how we recruited participants in §4.2. We discuss the phases of the experiment in §4.3 and detail the attack types in §4.4.

4.1 Simulation of User Interface Alternatives

The participants in our study were students enrolled in a university course that was *not* taught by one of the authors. Thus, we needed to be unintrusive so that the instructor need not be concerned about the impact of our study on students'

ability to reach essential course materials. Consequently, we could not control which computers students used to access the course web page. We could not provide students with a distinct physical TM, and we needed to support multiple browsers on multiple operating systems.

The most consequential change to the designs presented in §3 is that we implemented the TM as part of the course web page, not as a separate physical device. Rather, to simulate a separate physical device to the extent possible, we implemented the TM as a minimized window using the Google Web Toolkit. The minimized window appears as a title bar only. When the TM display is updated, the simulated TM beeps to attract the user's attention, as a real TM would. In addition, to account for the possibility that the student mutes the sound on her computer, the TM window's title bar flashes when it is updated (as a real TM's display could). The user must expand the TM window to view the TM's display. The expanded TM is shown in the context of Original in Figure 1(a). The elapsed time indicator gives the user definitive evidence that the information displayed on the TM is timely, i.e., is in response to the most recent SAS.

In addition to simulating the TM, we instrumented the course login web page to collect information about keystroke, mouse click, and focus events. This data was logged in a MySQL database that was the backend for the login web page. Information collected includes the type of event and a timestamp. For example, a user logging in with design Original and password `password` might induce the following sequence of log records: a focus event indicating that the password field is now focused; a keystroke `@`; another keystroke `@` (yielding `@@` — the SAS); a mouse click expanding the TM;[2] a blur event indicating that the password field lost focus; a focus event indicating that the password field is now focused; a keystroke `p`; a keystroke `a`; a keystroke `s`; ...; a keystroke `d`; a mouse click indicating that the user clicked the "log in" button; and a blur event indicating that the password field lost focus.

This logging design enables us to observe user activities that would not be evident from the final field contents upon form submission. For example, our logs include user mistakes, such as typing password characters and backspacing to remove them, which can be critical to accurately account for which password characters a user leaked to (simulated) malware on her machine (as measured in §5.2–§5.3.) The timestamps are primarily useful for ordering events, for computing login durations (see §5.1), and for determining when users have abandoned a session (e.g., walked away from their computer mid-login).

[2]Note that in a real implementation, the TM would be a separate physical device, and so this mouse click, the accompanying blur and the subsequent focus would not occur.

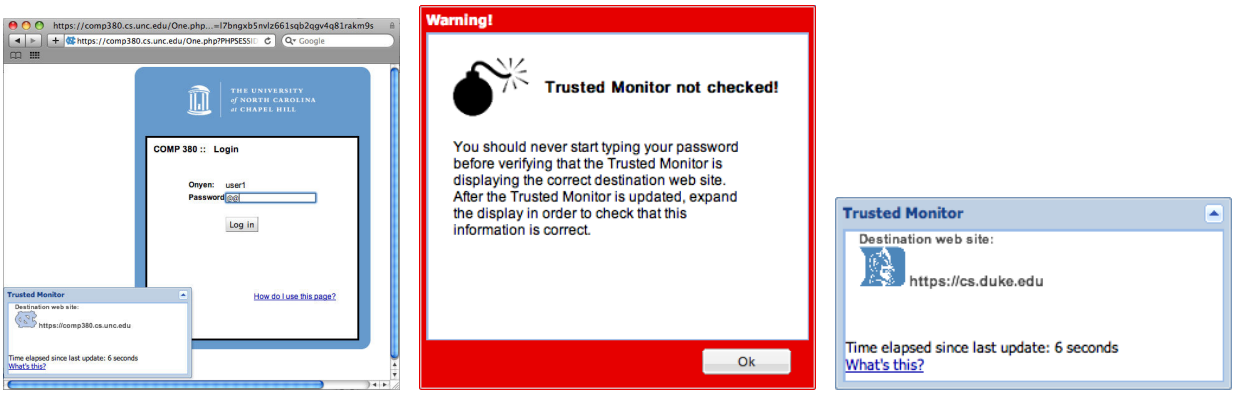
4.2 Participant Enrollment

The participants in our study were 85 students enrolled in *COMP 380: Computers and Society*, a one-semester course offered at the University of North Carolina (UNC) that is popular for satisfying general undergraduate requirements. There were 28 majors represented by the study participants, with the largest representations in Economics (14%), Business Administration (11%), Psychology (8%), Management and Society (6%), Biology (6%), and Journalism and Mass Communication (6%). One student was a Computer Science major, and there were no Engineering majors in the class. The breakdown of participants by year of study was: one freshman, one sophomore, 24 juniors, 58 seniors, and one graduate student. In the fall semester of 2009, there were three sections of the course offered, of sizes 45, 45 and 46 students. In these three sections, 27, 27, and 31 students opted in (yielding 85 total participants) while a total of 51 students chose not to opt in.

The participants were recruited through 20-minute presentations to each of the sections by one of the authors during the second week of class. These presentations covered: a very high-level discussion of the threats that malware pose and how Bumpy works; an explanation of the experiment; the users' rights as subjects in such an experiment; why the students might want to participate; and a brief demonstration of the Bumpy variants that participants from that section might see.

Students were motivated to enroll by the possibility of winning a cash award up to $150, the exact amount of which was determined by their performance in the experiment. We intended through the design of the award system to give students a financial incentive both to use the system regularly and to protect their passwords while doing so. In this award system, we reduced a student's possible winnings when she leaked password characters to our simulated attacks (except in the Control group, where the login provides no protection against the attacks), and we increased her possible winnings if she logged in frequently. Motivating the students financially to protect their passwords from our simulated attacks was necessary, as they had no other motivation to do so: since their chosen passwords were recorded in our logs so that we could determine when they leaked characters of their actual passwords — a fact made known to the students — our simulated attacks posed no actual threat to their (already known) passwords. Moreover, because we capped the amount of total money awarded across the whole class (students would be drawn at random and awarded their earned value until the cap was reached), the students were not motivated to assist each other. All of these points were made to the students during the 20-minute presentations to each course section.

Students opted in or out of the experiment upon access-



(a) Benign: Expanded TM. (b) Initial: TM not checked warning. (c) Attack: Wrong-Dest attack.

Figure 1. Screenshots with Bumpy-enabled login page from within the Safari browser.

ing the course web page after this presentation. Those who opted in filled out a background questionnaire prior to their first logins. According to their responses, the median computer usage among participants was 2-4 hours per day. After enrollment, there were 17 participants using Original, 16 using Graphical, 17 using NoTM, 19 using Challenge, and 16 using Control. None of these participants withdrew from using the system after the experiment commenced.

4.3 Experiment Phases

We conducted our experiment in four phases, to which each group of participants was subjected for the same durations and in the same order. Throughout these phases, each participant had access to an instructional video and help pages to explain how to log into the course web page (in benign circumstances) using the Bumpy design she was using. In addition, the help page explained that if the user detected what she thought might be an attack, she should reload the page. The four phases were as follows:

Initial (In): This phase lasted 15 days. During it, the users were not attacked and were provided automated instructions to walk them through how to log in during benign circumstances. For example, if a user in group Original typed "@@" in the password entry field, and then continued with typing her password without checking the TM (i.e., the TM was still minimized), then the user would be interrupted with instructions to check the TM before proceeding (see Figure 1(b)). In this phase, we provided as many such automated walk-through instructions as we could to ensure that the user logged in correctly.

Benign (Be): This phase followed Initial and lasted 28 days. It also had no simulated attacks, but the automated walk-through instructions for helping users log in were disabled.

Attack (At): This phase followed Benign and lasted 26 days. In this phase, we turned on the simulated attacks. Each login page included an Active attack (see §4.4) with probability 0.5; if the page was chosen to include an Active attack, the attack performed was chosen uniformly at random from among the Active attacks for that design. Automated instructions were still disabled in this phase. However, 12 days prior to starting this phase, we posted an announcement on the course web page to remind participants that if they encountered what they thought might be an attack, they simply need to reload the page and try again. The reloaded login page was attacked with the same probability as the original, i.e., our system did not treat reloaded pages any differently than the initial login page. We posted this announcement to avoid a situation where a user detected a problem but proceeded anyway since they did not know how to avoid it. Further, since we were controlling access to an essential course web page, we needed to ensure that students could always reach their course materials. Admittedly, reloading the page presents less of a hurdle to users than they would experience in practice to remedy malware they detect; this is a limitation of our experiment (see §6).

Attack-and-Warn (AW): This phase followed Attack and lasted 38 days. This phase was exactly like Attack, except that *after* a login attempt with a simulated attack and in which the user divulged password characters to the (simulated) malware, the system warned the user as to what actions she performed (or failed to perform) that caused the password characters to be divulged. These warnings thus served as a form of training to educate the users as to what they did wrong. We emphasize that in this phase, warnings appeared only *after* the login attempt was completed (in contrast to Initial, in which interruptions occurred during password entry). The only warnings that were suppressed from the user were for logins in which the user appeared to correct the error herself; e.g., she typed password characters

and then backspaced to remove them before properly utilizing the system. In such cases, we did not warn the user to avoid confusing her.

Figure 2 shows the login count per user during each of the four phases. Each box shows the first, second, and third quartiles; the whiskers cover points within 1.5 times the interquartile range. Outliers are shown as circles.

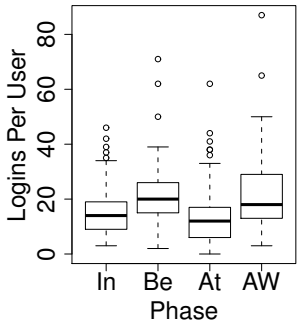


Figure 2. Logins per user in experiment phases.

Five days before the end of the Attack-and-Warn phase, the non-Control group participants were asked to complete an exit questionnaire. In order to remain eligible for compensation, the questionnaire had to be completed by the last day of this phase. Many of the questions were meant to assess the helpfulness of the warning messages from the Attack-and-Warn phase, allowing us to measure their training effect (see §5.3). Out of the 69 non-Control group students, 58 students completed the questionnaire.

4.4 Attacks

The simulated attacks to which we subjected participants in the Attack and Attack-and-Warn phases were of four types. These attack types were motivated by the threat model against which the Bumpy system was designed to protect, namely that the OS and its applications (e.g., web browser) are potentially under the control of malware, but that the Trusted Monitor (if any) and destination website remain uncompromised. The first three simulated attacks described below, namely Feigned-Fail, Wrong-Dest, and SAS-Present, are the Active attacks. In general, we counted password characters as leaked in a login if they would have been leaked to malware conducting the attack simulated in that login, given the natural implementation of the Bumpy variant in use. The fourth attack type is a Passive attack, described below.

Feigned-Fail (FF): These attacks simulated malware that interfered with Bumpy's operation. In the designs that employ a TM (Original, Graphical, and Challenge), this was implemented by simply not updating the TM, even after the participant performed the actions that should have caused it to be updated. We also implemented a Graphical variant in which the password-entry field would never turn green. In these cases, any password characters that the user entered were counted as leaked. In NoTM, which does not use a TM, this attack was simulated by failing to recognize the per-site SAS that the user entered (even if it was correct) and displaying an error message indicating this to the user. If the user then redefined the SAS through the `$...$` incantation (see §3.2.2), then all password characters typed after this were counted as leaked, since in a real implementation, malware could have caused the new SAS definition to apply to a site under its control, and hence captured the password. In a real Bumpy implementation, malware could mount these attacks, e.g., by simply blocking the Bumpy module's communication with the TM in designs Original, Graphical and Challenge, and by displaying the error message (and suppressing success indicators) in NoTM.

Wrong-Dest (WD): These attacks apply only to designs that use a TM (Original, Graphical, and Challenge). The simulated attack caused the wrong destination to be displayed on the TM. For our experiment, the destination displayed on the TM was `cs.duke.edu` and its graphical logo; Duke University is a nearby sports rival of UNC. An example TM display during this attack is shown in Figure 1(c). This behavior would be exhibited by a real implementation of Original, Graphical, or Challenge if malware attempted to redirect the forthcoming input to `cs.duke.edu`. Any password characters typed were counted as leaked.

SAS-Present (SP): These attacks tried to induce the user to enter her password without first entering her SAS, by providing a password entry field with the SAS already present. Thus, this attack applied only to designs Original, NoTM, and Challenge. In Original and Challenge, the password field was presented with "`@@`" already present. In NoTM, the SAS that that user previously specified for the course website was already present in the field when it was displayed. Though the SAS was already present, the TM (in Original and Challenge) was not updated, as malware would be unable to cause it to be updated in a real implementation. Any password characters the user typed prior to retyping the SAS were counted as leaked.

Passive (Pa): Any login during phases Attack or Attack-and-Warn that was not selected for an Active attack was instead considered subject to a passive attack, denoted Passive. Moreover, even a login selected for an Active attack but for which the user did not elicit any evidence of the chosen attack was instead included in Passive. Specifically, if in a login chosen for a Feigned-Fail or Wrong-Dest attack, the user never entered her SAS (in Original or Challenge) or toggled the password field green (in Graphical), then failure of the TM to update when it should (in a Feigned-Fail attack) or the appearance of `cs.duke.edu` (in a Wrong-Dest attack) would never be exhibited to the user. Such a login instance was labeled as Passive-attacked, instead. (SAS-Present attacks were always exhibited to the user, and so never counted as Passive.)

In a Passive-attacked login, any password characters typed prior to the update of the TM (induced by @@ in Original or Challenge, or toggling the password field to green in Graphical) or, in NoTM, prior to entry of @*str*@ (where *str* denotes any string), were considered leaked.

We do not claim that the Active attacks described above are a complete set of active attacks that malware could launch, though these include the most natural and subtle ones that we could envision and also require the specification of few additional parameters that would be subject to our own guesswork. In particular, we tried to devise attacks that users may misinterpret as typical discontinuities in the software experience, e.g., subtle changes between two versions of a particular application or website. More aggressive attacks could certainly be launched by displaying to users instructions that contradict what they should do, e.g., "We have upgraded to Bumpy 2.0. We now provide the SAS for you, so that you need not enter it." However, the exact form of such instructions may matter to their effectiveness. As such, we did not endeavor to explore the space of such active attacks. Given this, the primary parameter choice required by the above attacks is the selection of the destination in the Wrong-Dest attack. More subtle, phishing-style options could have been chosen (e.g., using `umc.edu`, which appears similar to UNC's domain (`unc.edu`), without changing the graphic logo), but we wanted to separate the participants' susceptibility to such phishing-style attacks, which are relevant outside the context of Bumpy as well as within it, from the utility of Bumpy itself.

5 Results of User Study

The experiment phases described in §4.3 provided an opportunity to study each Bumpy variant — Original (Or), Graphical (Gr), NoTM (No), and Challenge (Ch) — in multiple scenarios. Usability in benign scenarios, security in attack scenarios, and the effectiveness of warnings as a form of training are addressed in §5.1, §5.2, and §5.3, respectively. Also in §5.1, we compare the usability of each Bumpy variant to our control group, Control (Co), though Control is not considered in §5.2–5.3 since it does not offer resilience to the attacks for which Bumpy was designed (and thus that we tested in our attack experiments).

5.1 Usability in Benign Scenarios

To evaluate the usability of the Bumpy variants during the Benign phase, we measured the login success rate and duration, averaged over each user in each of the five groups.

Login success rate. We begin by discussing the login success rate. Let $success_{\mathsf{Be}}(u) = |L^{\mathsf{Su}}_{\mathsf{Be}}(u)|/|L_{\mathsf{Be}}(u)|$, where $L^{\mathsf{Su}}_{\mathsf{Be}}(u)$ and $L_{\mathsf{Be}}(u)$ denote the successful logins and all attempted logins, respectively, by user u in phase Benign. Then,

$$\bar{x}[success_{\mathsf{Be}}]^d = \operatorname*{avg}_{u \in U^d} success_{\mathsf{Be}}(u)$$
$$s[success_{\mathsf{Be}}]^d = \operatorname*{stddev}_{u \in U^d} success_{\mathsf{Be}}(u)$$

are the sample mean and standard deviation of login success rates for design-d users (denoted U^d) during the Benign phase. These measures are presented in Figure 3(a).

The sample mean approximates the corresponding population mean $\mu[success_{\mathsf{Be}}]^d$, and we use Analysis of Variance (ANOVA) to determine for which d, $e \in \{\mathsf{Original}, \mathsf{Graphical}, \mathsf{NoTM}, \mathsf{Challenge}, \mathsf{Control}\}$, $d \neq e$, we can invalidate the null hypothesis $\mu[success_{\mathsf{Be}}]^d = \mu[success_{\mathsf{Be}}]^e$ at a significance level of $\alpha = .05$.[3] ANOVA revealed no significant differences in the login success rates for the different designs, i.e., the extra steps required of users in our designs did not make it more difficult for users to log in successfully.

Login time. We now turn to measuring login duration for each of the Bumpy variants, i.e., the time required by users of each design to log into the course web page. For a login attempt ℓ, let $\mathsf{dur}(\ell)$ denote the time that transpired between the first action indicating that the user is commencing an attempt to log in — i.e., either the password field gaining focus, or the TM window being opened in one of the designs having a TM, whichever comes first — and the user clicking the "log in" button or pressing enter. Then,

$$duration_{\mathsf{Be}}(u) = \operatorname*{avg}_{\ell \in L^{\mathsf{Su}}_{\mathsf{Be}}(u)} \mathsf{dur}(\ell)$$

is the average time required by u to log in successfully in phase Benign. That said, when computing this average, we excluded any login in which there was a 15-second period of inactivity at any point between commencing (opening the TM window or focusing into the password field) and clicking the "log in" button (or pressing enter), considering such a login instance to be an outlier. This removed 5 out of 1650 logins in phase Benign, across all four Bumpy designs and Control. As with success rate,

$$\bar{x}[duration_{\mathsf{Be}}]^d = \operatorname*{avg}_{u \in U^d} duration_{\mathsf{Be}}(u)$$
$$s[duration_{\mathsf{Be}}]^d = \operatorname*{stddev}_{u \in U^d} duration_{\mathsf{Be}}(u)$$

denote the sample mean and standard deviation of the average successful login duration for users in U^d. These values are shown in Figure 3(b) for the different designs. The former approximates the population mean $\mu[duration_{\mathsf{Be}}]^d$, and

[3] We used Tukey's method of multiple comparisons [8] to test hypotheses involving Bumpy designs (Original, Graphical, NoTM, Challenge), and Dunnett's test [8] to test hypotheses involving Control.

d		
	Original	$.95 \pm .07$
	Graphical	$.93 \pm .09$
	NoTM	$.94 \pm .07$
	Challenge	$.89 \pm .14$
	Control	$.88 \pm .12$

(a) $\bar{x}[success_{\mathsf{Be}}]^d \pm s[success_{\mathsf{Be}}]^d$

d		
	Original	9.23 ± 2.04
	Graphical	8.59 ± 3.01
	NoTM	6.27 ± 1.89
	Challenge	11.51 ± 3.17
	Control	4.35 ± 1.90

(b) $\bar{x}[duration_{\mathsf{Be}}]^d \pm s[duration_{\mathsf{Be}}]^d$ in seconds

d \ e	Gr	No	Ch	Co
Or	.89	**.01**	**.05**	**.00**
Gr		.06	**.01**	**.00**
No			**.00**	.09
Ch				**.00**

(c) Hypothesis tests with null hypothesis $\mu[duration_{\mathsf{Be}}]^d = \mu[duration_{\mathsf{Be}}]^e$. Cells contain p-values; bold entries are results with significance $\alpha = .05$.

Figure 3. Benign **logins**

we again use ANOVA to determine whether we can invalidate the null hypothesis $\mu[duration_{\mathsf{Be}}]^d = \mu[duration_{\mathsf{Be}}]^e$ (for d, $e \in \{\mathsf{Original}, \mathsf{Graphical}, \mathsf{NoTM}, \mathsf{Challenge}, \mathsf{Control}\}$, $d \neq e$) at a significance level of $\alpha = .05$. The results are shown in Figure 3(c).

Not surprisingly, Control yielded significantly faster login times than the designs that utilized a TM (Original, Graphical, Challenge) — opening and inspecting the TM requires extra steps that take time. Moreover, Challenge was significantly slower than Original, Graphical, and NoTM, presumably owing to the additional diligence required to transcribe the challenge value from the TM into the password field. The only other significant result is that NoTM was faster than Original. Like Control, NoTM does not involve the use of a TM, and we believe that this is the most likely explanation for this result. An extra burden placed on users of NoTM is that of remembering their SAS, though according to the exit questionnaire only 2 out of the 14 participants who responded (out of 17 total assigned to NoTM) reported difficulty with this task. This seems to be reflected in the duration results since NoTM is competitive with Control in this respect.

Based on the sample means in Figure 3(b), it appears that the difference between the fastest (Control) and slowest (Challenge) methods is roughly 7 seconds in a successful login attempt. One might justify this additional delay on the basis of the greater security that Challenge offers, though other designs such as NoTM are much more competitive with Control in terms of login duration and have the possibility of offering the same protections as Challenge. As such, we now turn to measuring the actual protection that these designs offer, in an effort to further clarify the merits of each design.

5.2 Resilience in Attack Scenarios

The Attack phase of our study subjected users to simulated attacks, as might be carried out by malware that had infected the computer on which the user employed Bumpy or one of its variants; see §4.3 for a description of these simulated attacks. In this section we use the logins conducted during phase Attack to infer which of the designs Original, Graphical, NoTM and Challenge were most successful in enabling the user to protect her password. Specifically, for $a \in \{\mathsf{Feigned\text{-}Fail\ (FF)}, \mathsf{Wrong\text{-}Dest\ (WD)}, \mathsf{SAS\text{-}Present\ (SP)}, \mathsf{Passive\ (Pa)}\}$, let $L^a_{\mathsf{At}}(u)$ denote the logins by user u in phase Attack subject to attack a. Let $L^{\mathsf{Ac}}_{\mathsf{At}}(u) = L^{\mathsf{FF}}_{\mathsf{At}}(u) \cup L^{\mathsf{WD}}_{\mathsf{At}}(u) \cup L^{\mathsf{SP}}_{\mathsf{At}}(u)$ denote all Attack logins by u subject to an Active (Ac) attack, and let $L^{\mathsf{Any}}_{\mathsf{At}}(u) = L^{\mathsf{Ac}}_{\mathsf{At}}(u) \cup L^{\mathsf{Pa}}_{\mathsf{At}}(u)$ denote all Attack logins by u. Then, for any $a \in \{\mathsf{FF}, \mathsf{WD}, \mathsf{SP}, \mathsf{Ac}, \mathsf{Pa}, \mathsf{Any}\}$, let

$$avgLeaked^a_{\mathsf{At}}(u) = \operatorname*{avg}_{\ell \in L^a_{\mathsf{At}}(u)} \frac{\mathsf{leaked}(\ell)}{\mathsf{pwdLen}(\ell)} \quad (1)$$

$$maxLeaked^a_{\mathsf{At}}(u) = \max_{\ell \in L^a_{\mathsf{At}}(u)} \frac{\mathsf{leaked}(\ell)}{\mathsf{pwdLen}(\ell)} \quad (2)$$

where $\mathsf{pwdLen}(\ell)$ denotes the password length of user u at the time of login attempt ℓ (users were allowed to change their passwords during the experiment), and $\mathsf{leaked}(\ell)$ denotes the number of password characters that the user leaked to the (simulated) malware in login ℓ. Specifically, password characters were counted as leaked if they were typed while protection was not enabled, or if they occur in the password before other characters so leaked. (This conservative estimate assumed leakage up to and including any typed password characters.) So, $avgLeaked^a_{\mathsf{At}}(u)$ is the fraction of her password that user u leaked, on average, in logins of attack type a in phase Attack. Similarly, $maxLeaked^a_{\mathsf{At}}(u)$ is the cumulative fraction of her password that user u leaked across all logins of attack type a in phase Attack. As such, $avgLeaked^a_{\mathsf{At}}(u)$ indicates the fraction of u's password that an attacker might expect to obtain in a single login subjected to attack type a, and $maxLeaked^a_{\mathsf{At}}(u)$ is the fraction of u's password that an attacker might expect to gather over an extended period of observing the user type her password in logins of attack type a.

The results for the various designs $d \in \{\mathsf{Original},$

Graphical, NoTM, Challenge} are shown in Figure 4. Specifically, Figure 4(a) lists:

$$\bar{x}[avgLeaked^a_{\mathsf{At}}]^d = \operatorname*{avg}_{u \in U^d} avgLeaked^a_{\mathsf{At}}(u)$$
$$s[avgLeaked^a_{\mathsf{At}}]^d = \operatorname*{stddev}_{u \in U^d} avgLeaked^a_{\mathsf{At}}(u)$$

and Figure 4(b) lists:

$$\bar{x}[maxLeaked^a_{\mathsf{At}}]^d = \operatorname*{avg}_{u \in U^d} maxLeaked^a_{\mathsf{At}}(u)$$
$$s[maxLeaked^a_{\mathsf{At}}]^d = \operatorname*{stddev}_{u \in U^d} maxLeaked^a_{\mathsf{At}}(u)$$

For designs $d, e \in$ {Original, Graphical, NoTM, Challenge}, $d \neq e$, and for attack type a, we can use this data to test the null hypotheses $\mu[avgLeaked^a_{\mathsf{At}}]^d = \mu[avgLeaked^a_{\mathsf{At}}]^e$ and $\mu[maxLeaked^a_{\mathsf{At}}]^d = \mu[maxLeaked^a_{\mathsf{At}}]^e$ where $\mu[avgLeaked^a_{\mathsf{At}}]^d$ and $\mu[maxLeaked^a_{\mathsf{At}}]^d$ denote the population means that correspond to the sample means $\bar{x}[avgLeaked^a_{\mathsf{At}}]^d$ and $\bar{x}[maxLeaked^a_{\mathsf{At}}]^d$, respectively. Upon conducting these tests, the hypotheses that were rejected at significance level $\alpha = .05$ are

$$\mu[avgLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{No}} = \mu[avgLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{Or}} \quad (3)$$
$$\mu[maxLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{No}} = \mu[maxLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{Or}} \quad (4)$$
$$\mu[avgLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{No}} = \mu[avgLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{Ch}} \quad (5)$$
$$\mu[maxLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{No}} = \mu[maxLeaked^{\mathsf{SP}}_{\mathsf{At}}]^{\mathsf{Ch}} \quad (6)$$

That is, NoTM performed significantly worse than Original and Challenge in defending against SAS-Present attacks, in both the $avgLeaked^{\mathsf{SP}}_{\mathsf{At}}()$ and $maxLeaked^{\mathsf{SP}}_{\mathsf{At}}()$ measures. We believe this can be explained by the fact that Original and Challenge provide feedback that the user expects to see prior to entering her password, namely the information displayed on the TM. Indeed it is *necessary* for the user to receive this feedback to continue the login process in the Challenge case. We believe that this feedback explains why Original and Challenge outperform NoTM in this attack, and why Challenge offers the lowest sample mean $\bar{x}[maxLeaked^{\mathsf{SP}}_{\mathsf{At}}]^d = .13$ (i.e., provides the best protection for the user's password) among all designs d.

An additional noteworthy observation from Figure 4 is that the Bumpy variants we tested *do* improve users' abilities to protect their passwords. Today's web password logins do not protect at all against even Passive attacks in our threat model (i.e., malware on the client's machine); i.e., if included in the tables in Figure 4, today's logins would be listed as 1.0 ± 0.0 in the Passive column. In contrast, all of our designs performed convincingly better against Passive attacks in the average case (Figure 4(a)), and Graphical and Challenge performed well against Passive attacks even when considering the case in which an attacker monitors the user over an extended period of time (represented by Figure 4(b)). While the low password leakage under Passive attacks shows that the compliance rate for secure attention sequences was high, we wanted to know if the users were comfortable with this model for wider usage. When asked if they would be happy to use this system for other websites, 62% of the 58 (non-Control group) exit questionnaire respondents agreed, 14% were neutral, and 24% disagreed. While in the Attack phase users leaked more of their passwords during Active attacks than during Passive attacks, §5.3 presents evidence that training via warnings in the Attack-and-Warn phase was effective in reducing this leakage in most cases.

As part of the exit questionnaire, users were asked whether they successfully noticed and avoided an attack on their password during this phase. In order to test the null hypothesis that the user responses and true performance are not correlated, we compared the binary response variable with actual behavior (a value of 1 indicating that they avoided at least one attack). We observed a Pearson correlation coefficient [21] of $\rho = .24$ (p-value $= .07$). While this p-value does not allow us to reject the null hypothesis with $\alpha = .05$, the positive ρ suggests that users who responded "yes" tended to avoid at least one attack during this phase.

5.3 Effects of Training for Attack Scenarios

The last phase of our study is Attack-and-Warn (recall §4.3). Users were both attacked as in Attack and then warned *after* completion of the login attempt if they leaked any password characters and, if so, informed why the characters were leaked. This provided a form of training. Our last tests focus on whether this training was effective at improving users' abilities to protect their passwords.

Let $L^a_{\mathsf{AW}}(u)$ denote the logins by user u in Attack-and-Warn categorized as attack type $a \in$ {Passive, Feigned-Fail, Wrong-Dest, SAS-Present}. We additionally define two subsets of the logins $L^a_{\mathsf{AW}}(u)$, namely those logins that occurred *before* the first warning (if any) was presented to u for attack type a, denoted $B^a_{\mathsf{AW}}(u)$, and those logins that occurred *after* the first warning (if any) for a was presented to u, denoted $A^a_{\mathsf{AW}}(u)$. The login attempt that caused the first warning for attack a to be presented to u, if any, is defined to be in $B^a_{\mathsf{AW}}(u)$. Note that if u was presented with no warnings for attack a for the entirety of Attack-and-Warn, then $B^a_{\mathsf{AW}}(u)$ and $A^a_{\mathsf{AW}}(u)$ are both empty. We can now go on to define these sets for the Active and Any attacks in the natural way. Specifically, $L^{\mathsf{Ac}}_{\mathsf{AW}}(u) = L^{\mathsf{FF}}_{\mathsf{AW}}(u) \cup L^{\mathsf{WD}}_{\mathsf{AW}}(u) \cup L^{\mathsf{SP}}_{\mathsf{AW}}(u)$; $B^{\mathsf{Ac}}_{\mathsf{AW}}(u) = B^{\mathsf{FF}}_{\mathsf{AW}}(u) \cup B^{\mathsf{WD}}_{\mathsf{AW}}(u) \cup B^{\mathsf{SP}}_{\mathsf{AW}}(u)$; and $A^{\mathsf{Ac}}_{\mathsf{AW}}(u) = A^{\mathsf{FF}}_{\mathsf{AW}}(u) \cup A^{\mathsf{WD}}_{\mathsf{AW}}(u) \cup A^{\mathsf{SP}}_{\mathsf{AW}}(u)$. Similarly, $L^{\mathsf{Any}}_{\mathsf{AW}}(u) = L^{\mathsf{Pa}}_{\mathsf{AW}}(u) \cup L^{\mathsf{Ac}}_{\mathsf{AW}}(u)$; $B^{\mathsf{Any}}_{\mathsf{AW}}(u) = B^{\mathsf{Pa}}_{\mathsf{AW}}(u) \cup B^{\mathsf{Ac}}_{\mathsf{AW}}(u)$; and $A^{\mathsf{Any}}_{\mathsf{AW}}(u) = A^{\mathsf{Pa}}_{\mathsf{AW}}(u) \cup A^{\mathsf{Ac}}_{\mathsf{AW}}(u)$.

d \ a	Passive	Active	Feigned-Fail	Wrong-Dest	SAS-Present	Any
Original	.04 ± .07	.18 ± .27	.32 ± .45	.25 ± .50	.05 ± .12	.13 ± .17
Graphical	.03 ± .11	.41 ± .35	.39 ± .33	.66 ± .51	n/a	.17 ± .19
NoTM	.12 ± .28	.43 ± .41	.17 ± .34	n/a	.53 ± .47	.25 ± .30
Challenge	.06 ± .24	.25 ± .37	.21 ± .36	.43 ± .53	.06 ± .18	.14 ± .25

(a) $\bar{x}[avgLeaked^a_{\mathsf{At}}]^d \pm s[avgLeaked^a_{\mathsf{At}}]^d$

d \ a	Passive	Active	Feigned-Fail	Wrong-Dest	SAS-Present	Any
Original	.28 ± .41	.33 ± .49	.40 ± .52	.25 ± .50	.14 ± .36	.49 ± .48
Graphical	.13 ± .35	.77 ± .42	.77 ± .42	.67 ± .52	n/a	.67 ± .47
NoTM	.23 ± .43	.71 ± .47	.34 ± .46	n/a	.69 ± .48	.71 ± .47
Challenge	.09 ± .28	.44 ± .51	.40 ± .49	.43 ± .53	.13 ± .34	.43 ± .50

(b) $\bar{x}[maxLeaked^a_{\mathsf{At}}]^d \pm s[maxLeaked^a_{\mathsf{At}}]^d$

Figure 4. Sample statistics for Attack logins.

We begin our analysis of the effects of warnings by computing $avgLeaked^a_{\mathsf{AW}}(u)$ and $maxLeaked^a_{\mathsf{AW}}(u)$ for different attacks a and users u. These are computed analogously to equations (1)–(2), specifically by replacing $L^a_{\mathsf{At}}(u)$ with $L^a_{\mathsf{AW}}(u) \setminus B^a_{\mathsf{AW}}(u)$. That is, the logins $B^a_{\mathsf{AW}}(u)$ are removed in the calculation of $avgLeaked^a_{\mathsf{AW}}(u)$ and $maxLeaked^a_{\mathsf{AW}}(u)$ since these logins cannot exhibit the effects of the warnings in phase Attack-and-Warn. We then compute sample statistics $\bar{x}[avgLeaked^a_{\mathsf{AW}}]^d$ and $s[avgLeaked^a_{\mathsf{AW}}]^d$ over all users $u \in U^d$ where $L^a_{\mathsf{AW}}(u) \setminus B^a_{\mathsf{AW}}(u) \neq \emptyset$, and similarly for $\bar{x}[maxLeaked^a_{\mathsf{AW}}]^d$ and $s[maxLeaked^a_{\mathsf{AW}}]^d$. The results are shown in Figure 5.

The results in Figure 5 suggest in some ways that the warnings did serve a training purpose. For example, $\bar{x}[avgLeaked^{\mathsf{Ac}}_{\mathsf{AW}}]^d < \bar{x}[avgLeaked^{\mathsf{Ac}}_{\mathsf{At}}]^d$ for all designs d, suggesting that defense against Active attacks generally improved. For the Challenge design, moreover, $\bar{x}[avgLeaked^a_{\mathsf{AW}}]^{\mathsf{Ch}} < \bar{x}[avgLeaked^a_{\mathsf{At}}]^{\mathsf{Ch}}$ for all Active attacks a (and remained essentially unchanged for Passive attacks), suggesting that this design benefited from warnings as a form of training. To examine the effectiveness of these warnings more carefully, we define

$$avgBin^a_{\mathsf{At}}(u) = \begin{cases} \mathsf{none} & \text{if } avgLeaked^a_{\mathsf{At}}(u) = 0 \\ \mathsf{some} & \text{if } avgLeaked^a_{\mathsf{At}}(u) \in (0, .5) \\ \mathsf{most} & \text{if } avgLeaked^a_{\mathsf{At}}(u) \in [.5, 1] \end{cases}$$

$$maxBin^a_{\mathsf{At}}(u) = \begin{cases} \mathsf{none} & \text{if } maxLeaked^a_{\mathsf{At}}(u) = 0 \\ \mathsf{some} & \text{if } maxLeaked^a_{\mathsf{At}}(u) \in (0, .5) \\ \mathsf{most} & \text{if } maxLeaked^a_{\mathsf{At}}(u) \in [.5, 1] \end{cases}$$

and define $avgBin^a_{\mathsf{AW}}(u)$ and $maxBin^a_{\mathsf{AW}}(u)$ analogously. Now, let $y[avgBin^a_{\mathsf{At}}]^d$ (respectively, $y[maxBin^a_{\mathsf{At}}]^d$, $y[avgBin^a_{\mathsf{AW}}]^d$, $y[maxBin^a_{\mathsf{AW}}]^d$) denote a random variable that takes on $avgBin^a_{\mathsf{At}}(u)$ (respectively, $maxBin^a_{\mathsf{At}}(u)$, $avgBin^a_{\mathsf{AW}}(u)$, $maxBin^a_{\mathsf{AW}}(u)$) under random choice of $u \in U^d$, and let $\nu[avgBin^a_{\mathsf{At}}]^d$ (respectively, $\nu[maxBin^a_{\mathsf{At}}]^d$, $\nu[avgBin^a_{\mathsf{AW}}]^d$, $\nu[maxBin^a_{\mathsf{AW}}]^d$) denote the corresponding population random variable. Then, for some designs d, we can reject the null hypothesis $\nu[avgBin^a_{\mathsf{At}}]^d \sim \nu[avgBin^a_{\mathsf{AW}}]^d$ (i.e., that these random variables are distributed identically), or $\nu[maxBin^a_{\mathsf{At}}]^d \sim \nu[maxBin^a_{\mathsf{AW}}]^d$, at significance $\alpha = .05$,[4] meaning that training helped in a statistically significant way.[5] Figure 5 includes p-values for these tests in parentheses, with those for significant results in boldface. That said, there are plenty of examples in Figure 5 yielding inconclusive results.

To the extent that warnings did serve a training purpose, they did not entirely overcome some of the challenges that the Attack phase evidenced in some designs. Recall from §5.2 that NoTM was statistically inferior to Original and Challenge against SAS-Present attacks; see rejected hypotheses (3)–(6). It appears that the weaknesses of NoTM persisted to some extent even after warnings as a form of training, in that the following hypotheses could be rejected at significance level $\alpha = .05$:

$$\mu[avgLeaked^{\mathsf{Ac}}_{\mathsf{AW}}]^{\mathsf{No}} = \mu[avgLeaked^{\mathsf{Ac}}_{\mathsf{AW}}]^{\mathsf{Ch}} \quad (7)$$

$$\mu[maxLeaked^{\mathsf{Pa}}_{\mathsf{AW}}]^{\mathsf{No}} = \mu[maxLeaked^{\mathsf{Pa}}_{\mathsf{AW}}]^{\mathsf{Gr}} \quad (8)$$

[4] These hypotheses were tested using a multinomial ordered logit model and included only users $u \in U^d$ for which $L^a_{\mathsf{At}}(u) \neq \emptyset$ and $L^a_{\mathsf{AW}}(u) \neq \emptyset$, since otherwise we would include users who did not face attack a in either the Attack or Attack-and-Warn phase. Note that this test can return $\perp$ if all samples from one distribution are the same or if the samples from one distribution are the same as those from the other.

[5] The rejected null hypotheses $\nu[maxBin^{\mathsf{Pa}}_{\mathsf{At}}]^{\mathsf{No}} \sim \nu[maxBin^{\mathsf{Pa}}_{\mathsf{AW}}]^{\mathsf{No}}$ is an exception, in that $\bar{x}[maxLeaked^{\mathsf{Pa}}_{\mathsf{At}}]^{\mathsf{No}} < \bar{x}[maxLeaked^{\mathsf{Pa}}_{\mathsf{AW}}]^{\mathsf{No}}$ suggests that NoTM performed significantly *worse* in this case. Possible reasons for this are discussed below.

d	a: Passive	Active	Feigned-Fail	Wrong-Dest	SAS-Present	Any
Original	.04 ± .06 (1.00)	.12 ± .22 (.91)	.07 ± .16 (.17)	.29 ± .46 (⊥)	.02 ± .05 (1.00)	.06 ± .09 (.87)
Graphical	.00 ± .00 (⊥)	.18 ± .24 (**.01**)	.23 ± .33 (**.01**)	.12 ± .30 (**.05**)	n/a	.06 ± .10 (.10)
NoTM	.09 ± .16 (**.03**)	.29 ± .39 (.10)	.32 ± .44 (.40)	n/a	.15 ± .33 (**.04**)	.18 ± .23 (.75)
Challenge	.07 ± .23 (.08)	.04 ± .09 (.31)	.12 ± .31 (.57)	.07 ± .15 (.80)	.00 ± .00 (⊥)	.08 ± .23 (.55)

(a) $\bar{x}[avgLeaked^a_{\mathsf{AW}}]^d \pm s[avgLeaked^a_{\mathsf{AW}}]^d$ (p-value for test of null hypothesis that $\nu[avgBin^a_{\mathsf{At}}]^d \sim \nu[avgBin^a_{\mathsf{AW}}]^d$)

d	a: Passive	Active	Feigned-Fail	Wrong-Dest	SAS-Present	Any
Original	.32 ± .46 (1.00)	.36 ± .47 (.76)	.18 ± .40 (.30)	.31 ± .46 (⊥)	.10 ± .29 (1.00)	.56 ± .47 (.65)
Graphical	.00 ± .00 (⊥)	.51 ± .51 (**.03**)	.51 ± .51 (**.03**)	.22 ± .42 (**.05**)	n/a	.45 ± .51 (.10)
NoTM	.49 ± .47 (**.02**)	.44 ± .51 (**.03**)	.38 ± .50 (.85)	n/a	.25 ± .45 (**.05**)	.68 ± .46 (1.00)
Challenge	.20 ± .36 (.07)	.21 ± .40 (.56)	.18 ± .40 (.65)	.19 ± .34 (1.00)	.00 ± .00 (**.00**)	.30 ± .43 (.65)

(b) $\bar{x}[maxLeaked^a_{\mathsf{AW}}]^d \pm s[maxLeaked^a_{\mathsf{AW}}]^d$ (p-value for test of null hypothesis that $\nu[maxBin^a_{\mathsf{At}}]^d \sim \nu[maxBin^a_{\mathsf{AW}}]^d$)

Figure 5. Sample statistics for Attack-and-Warn logins. Bold p-values indicate significance $\alpha = .05$.

That is, NoTM was significantly inferior to Challenge against Active attacks (in terms of $avgLeaked^{\mathsf{Ac}}_{\mathsf{AW}}()$) and to Graphical against Passive attacks (in terms of $maxLeaked^{\mathsf{Pa}}_{\mathsf{AW}}()$). The generally non-trivial password leakage with NoTM, even after training by warnings, ran counter to our expectations (see §3.2.2).

To better illustrate the impact of warnings, in Figure 6 we show a histogram per attack type a and design d that illustrates the improvement of users $u \in U^d$, specifically the distributions of

$$avgImprove^a_{\mathsf{AW}}(u) = \frac{avgLeaked^a_{\mathsf{At}}(u) - avgLeaked^a_{\mathsf{AW}}(u)}{avgLeaked^a_{\mathsf{At}}(u)}$$

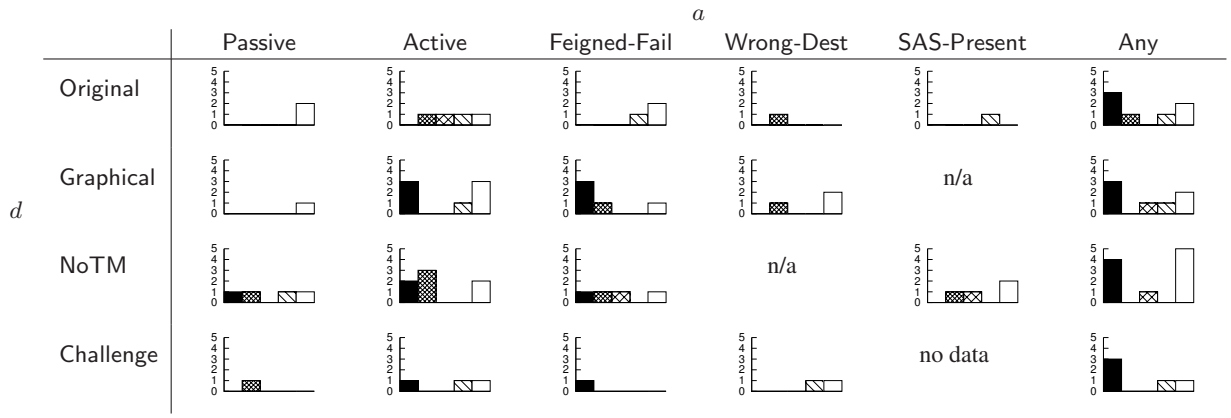
$$maxImprove^a_{\mathsf{AW}}(u) = \frac{maxLeaked^a_{\mathsf{At}}(u) - maxLeaked^a_{\mathsf{AW}}(u)}{maxLeaked^a_{\mathsf{At}}(u)}$$

These measures capture each user's absolute improvement relative to the improvement possible for that user. Figure 6 is limited to those users u who actually received warnings for a (i.e., $A^a_{\mathsf{AW}}(u) \neq \emptyset$), and for whom $L^a_{\mathsf{At}}(u) \neq \emptyset$ and $avgLeaked^a_{\mathsf{At}}(u) \neq 0$. Otherwise, we cannot compute $avgImprove^a_{\mathsf{AW}}(u)$ or $maxImprove^a_{\mathsf{AW}}(u)$.

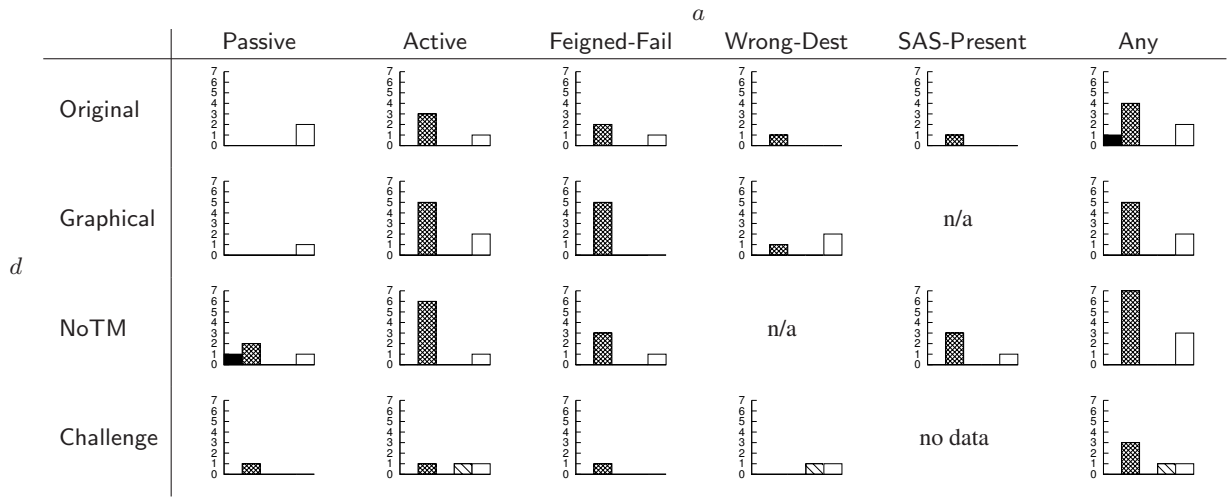
One interesting observation from Figure 6(a) is that there are several cases in which users performed worse during Attack-and-Warn than they did in Attack, at least judging from $avgImprove^a_{\mathsf{AW}}(u)$. This is evidenced in Figure 6(a) by black bars flush against the y-axis. This bar indicates the number of users for which $avgImprove^a_{\mathsf{AW}}(u) < 0$. (And recall that these histograms show only users for which $avgLeaked^a_{\mathsf{At}}(u) > 0$; some users with $avgLeaked^a_{\mathsf{At}}(u) = 0$ performed worse in Attack-and-Warn, as well.) While these users were by no means in a majority, in some cases they performed substantially worse in Attack-and-Warn than they did in Attack. These poor showings in Attack-and-Warn result in some values of $\bar{x}[avgLeaked^a_{\mathsf{AW}}]^d$ increasing slightly from $\bar{x}[avgLeaked^a_{\mathsf{At}}]^d$; c.f., Figures 4(a) and 5(a). The underlying reason why these users did more poorly in Attack-and-Warn than in Attack is unknown to us, but two possibilities come to mind. One is that these users misunderstood our warnings in a way that caused them to perform more poorly. A second possibility is simply carelessness or ambivalence on the part of the user, which seems likely given that Attack-and-Warn commenced 70 days after our recruiting presentations that outlined their incentives for the experiment.

Another interesting observation is the bimodality that is evident in Figure 6(b). This shows that with few exceptions, those that had room to improve either improved greatly ($maxImprove^a_{\mathsf{AW}}(u) \in [.75, 1.0]$, corresponding to the rightmost, white bars in the histograms) or very little ($maxImprove^a_{\mathsf{AW}}(u) \in [0, .25)$, corresponding to the dark gray bars one bar-width from the y-axis). Unfortunately it appears that the latter was more common, but in some sense this is unsurprising: even a *single* login instance in which the user u mistakenly leaked her password results in a value of $maxLeaked^a_{\mathsf{AW}}(u)$ that cannot be offset with numerous mistake-free logins. In this respect, we are encouraged by the users with large values for $maxImprove^a_{\mathsf{AW}}(u)$.

Reflecting on our results, a lesson is that training in a real deployment should perhaps require users to repeat logins that would have leaked password characters to malware had it been present. Instead, in Attack-and-Warn, we simply warned users but allowed them to proceed to the course web page if they entered the correct password. We considered forcing users to repeat logins in which they leaked characters but feared that such heavy-handed training would



(a) Distribution of $avgImprove^{a}_{\mathsf{AW}}(u)$



(b) Distribution of $maxImprove^{a}_{\mathsf{AW}}(u)$

Figure 6. Histograms showing distribution of $avgImprove^{a}_{\mathsf{AW}}(u)$ and $maxImprove^{a}_{\mathsf{AW}}(u)$ for users $u \in U^d$ such that $A^{a}_{\mathsf{AW}}(u) \neq \emptyset$, $L^{a}_{\mathsf{At}}(u) \neq \emptyset$, and $avgLeaked^{a}_{\mathsf{At}}(u) \neq 0$. Each histogram includes five bars (some potentially of height zero), corresponding to ranges $(-\infty, 0)$ (leftmost, darkest bar), $[0, .25)$, $[.25, .5)$, $[.5, .75)$, and $[.75, 1.0]$ (rightmost, lightest bar). The y-axis is the number of users.

be viewed as an obstruction, potentially causing participants to withdraw or the course instructor to protest. In a real deployment in a security-conscious organization, however, such an approach might be warranted.

To further measure the training effect of the warning messages, we looked to the exit questionnaire. There was a question corresponding to each individual type of attack (varying by design) which asked if the warning message helped them to avoid falling for the same attack in the future. A screenshot of the warning message was presented along with a five-level Likert scale ranging from 1 (Strongly Disagree) to 5 (Strongly Agree) and an additional sixth choice for users who reported not reading the warning message. In order to test the null hypothesis that the user responses and true performance are not correlated, we compared the responses with actual behavior by looking at the number of times the user saw the warning message in question. First we removed all users who never saw the warning message, and then we removed users who did not read the warning message. On the remaining data, we observed a Pearson correlation coefficient of $\rho = -.20$ (p-value $= .10$). While not statistically significant, the neg-

ative ρ suggests that the more helpful a user found a warning message, the fewer times they fell for the corresponding attack.

6 Discussion and Conclusions

Specific study conclusions. Our study suggests several take-away messages about the various Bumpy designs tested here.

1. The NoTM design is very attractive for both deployment (since it uses no TM) and user login duration (see Figure 3(b)). Unfortunately (and contrary to our expectations, see §3.2.2) there is significant evidence that it is more susceptible to users leaking password characters than other designs (see rejected hypotheses (3)–(6)). This holds even after training via warnings (see rejected hypotheses (7)–(8)), even though warnings were effective in significantly improving NoTM's security against some attacks (see Figure 5). As such, if NoTM is deployed, it warrants a concerted training effort for users.
2. The Challenge design offers generally low password leakage after training via warnings, at least judging from the sample means illustrated in Figure 5. In particular, the sample means of Challenge were the lowest across designs for all Active attacks generally, the only exception being Feigned-Fail in Figure 5(a). While the evidence supporting the greater security of Challenge in comparison to other designs was statistically significant only in some cases (see rejected hypotheses (5)–(6) and (7)), it appears to be a generally good choice for security. This is a testament to *requiring* users to examine the TM in the login process, though it does result in statistically significantly longer login times (see Figure 3(b)).
3. The evidence for a move to a graphical design like Graphical is weak. Graphical offered no significant improvements over Original in login success rate, duration, or password leakage in either Attack or Attack-and-Warn. While it did exhibit significant improvement due to warnings in Attack-and-Warn (see Figure 5), the attacks for which it did so often still captured as many password characters as with Original or Challenge, in terms of sample means. The best argument we see for Graphical is that it yielded the least leakage (in terms of sample means) for Passive attacks (Figures 4, 5), but given the implementation challenges that Graphical introduces (see §3.2.1), we believe the evidence of its benefits would need to be stronger to advocate for it.

Broader observations. Aside from the above conclusions, we provide more general observations that extend beyond the specific designs studied here, albeit with a degree of speculation.

1. Users appear to readily adapt to employing secure attention sequences (at least generic ones, versus site-specific ones), as evidenced by the low password leakage versus Passive attacks in our designs that employ one. Techniques that leverage secure attention sequences for useful properties thus hold promise.
2. Though the results of our study suggest a tradeoff between login duration and security, we see no reason to conclude that this tradeoff is fundamental. Rather, we consider it a fascinating open problem to design a login system that offers both the speed to which users are accustomed today and security against the attacks we consider here.
3. The additional security offered by Challenge suggests that interactive security indicators yield better security than ones that a user is asked to simply observe. This is a direction that deserves further attention, in our opinion.
4. That some users were unfazed by warnings in Attack-and-Warn suggests that to mold user behavior, training that requires a user to repeat the task at hand immediately following a mistake would be warranted. As this model can potentially disrupt the user's primary activity, though, it is perhaps more acceptable to include such training as a separate user activity. Automated reinforcement during teachable moments, e.g., when a user makes a mistake during normal operation, may also be desirable, but such designs must be weighed against the ability of attackers to use such mechanisms maliciously. For example, attackers may provide malicious instructions that confuse users.
5. Many of our Active attacks were designed to mimic changes of behavior that might seem familiar to users, owing to relatively frequent discontinuities that pervade users' software experiences (failures, software updates changing behavior in subtle ways, etc.). Based on our study, it appears that many users are unprepared to distinguish between a benign discontinuity and a subtle attack without training.

Limitations. Our study did not attempt to evaluate the usability of Bumpy designs for protecting multiple secret input fields on a single web form, input fields to multiple websites, or secrets of greater complexity or length than a typical password. (We simply enforced a minimum length requirement of 8 characters and placed no requirements as to the "strength" of the password chosen.) More complex secrets, in particular, may be problematic for usability since Bumpy prevents these keystrokes from echoing to the user's display. Regarding forgotten passwords, our study did not employ Bumpy protections on the "reset password" page.

As discussed previously, the step required of users who did detect malware (reloading the page) and our simulation of the TM are both sources of unrealism in our experiment, albeit unavoidable ones given the constraints of the course

setting. While a financial incentive was necessary to motivate our users to protect their passwords, such incentives are not so directly available in practice and could have had unintended effects on user behavior.

As discussed in §4.4, we do not claim that our simulated attacks are exhaustive. For example, we did not consider phishing attacks in our study (though previous studies have shown promise in educating users to detect them through a training regimen similar to our Attack-and-Warn phase [28]). Nevertheless, the attacks we did simulate allowed us to draw several conclusions about Bumpy and more broadly, as discussed above. At the same time, the numerous conditions required for computing some of our measures — e.g., that $L^a_{\mathsf{At}}(u) \neq \emptyset$, $A^a_{\mathsf{AW}}(u) \neq \emptyset$, and $avgLeaked^a_{\mathsf{At}}(u) \neq 0$ in order to compute $avgImprove^a_{\mathsf{AW}}(u)$ — reduced the number of sample points (users) for some of our measures to an uncomfortably small number. If, after further investigation, the attacks we considered do seem to be the primary threats, it may be desirable to perform larger studies focused on each attack.

Acknowledgements

We are particularly grateful to the students of COMP380 at UNC in the Fall 2009 semester, and to the course instructor Tessa Joseph Nicholas, for permitting us to conduct this experiment. We are also especially grateful to Chris Wiesen of the Odum Institute at UNC, for his guidance on the statistical evaluations in this paper. We thank Kathleen McCune for her assistance on the study questionnaire and Darrell Bethea, Ting-Fang Yen, and the anonymous reviewers for comments on previous versions of this paper. This work was supported in part by NSF grant CT-0756998 and by grant DAAD19-02-1-0389 from the Army Research Office.

References

[1] D. Balfanz and E. W. Felten. Hand-held computers can be better smart cards. In *USENIX Security*, Aug. 1999.

[2] K. Borders and A. Prakash. Securing network input via a trusted input proxy. In *USENIX HotSec*, Aug. 2007.

[3] S. Chiasson, P. C. Van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *USENIX Security*, Aug. 2006.

[4] D. E. Clarke, B. Gassend, T. Kotwal, M. Burnside, M. van Dijk, S. Devadas, and R. L. Rivest. The untrusted computer problem and camera-based authentication. In *International Conf. Pervasive Computing*, 2002.

[5] P. Coogan. Zeus, king of the underground crimeware toolkits. Symantec Blogs, Aug. 2009.

[6] L. F. Cranor. What do they "indicate"?: evaluating security and privacy indicators. *Interactions*, 13(3):45–47, 2006.

[7] L. Duflot, O. Levillain, B. Morin, and O. Grumelard. Getting into the SMRAM: SMM reloaded. Central Directorate for Information Systems Security, 2009.

[8] L. K. Edwards. *Applied Analysis of Variance in Behavioral Science*. CRC Press, 1993.

[9] S. Egelman, L. F. Cranor, and J. Hong. You've been warned: An empirical study of the effectiveness of web browser phishing warnings. In *CHI*, 2008.

[10] E. Gabber, P. B. Gibbons, D. M. Kristol, Y. Matias, and A. Mayer. On secure and pseudonymous client-relationships with multiple servers. *ACM TISSEC*, 2:390–415, Nov. 1999.

[11] S. Garriss, R. Cáceres, S. Berger, R. Sailer, L. van Doorn, and X. Zhang. Trustworthy and personalized computing on public kiosks. In *MobiSys*, June 2008.

[12] D. Grawrock. *Dynamics of a Trusted Platform: A Building Block Approach.* Intel Press, 2008.

[13] IBM Zurich Research Lab. Security on a stick. Press release, Oct. 2008.

[14] M. Kassner. Carberp: Quietly replacing Zeus as the financial malware of choice. TechRepublic IT Security Blogs, Oct. 2010.

[15] S. T. King, P. M. Chen, Y.-M. Wang, C. Verbowski, H. J. Wang, and J. R. Lorch. SubVirt: Implementing malware with virtual machines. In *IEEE Symp. Security and Privacy*, 2006.

[16] J. M. McCune, B. Parno, A. Perrig, M. K. Reiter, and H. Isozaki. Flicker: An execution infrastructure for TCB minimization. In *ACM EuroSys*, Apr. 2008.

[17] J. M. McCune, A. Perrig, and M. K. Reiter. Safe passage for passwords and other sensitive data. In *ISOC NDSS*, Feb. 2009.

[18] B. A. Myers. Using handhelds and PCs together. *CACM*, 44(11), Nov. 2001.

[19] A. Oprea, D. Balfanz, G. Durfee, and D. K. Smetters. Securing a remote terminal application with a mobile trusted device. In *ACSAC*, 2004.

[20] Organisation for Economic Co-operation and Development (OECD). Malicious software (malware): A threat to the internet economy. Technical Report DSTI/ICCP/REG(2007)5/FINAL, OECD, June 2008.

[21] D. Rees. *Foundations of Statistics*. Chapman and Hall/CRC, 1987.

[22] B. Ross, C. Jackson, N. Miyake, D. Boneh, and J. C. Mitchell. Stronger password authentication using browser extensions. In *USENIX Security*, Aug. 2005.

[23] S. J. Ross, J. L. Hill, M. Y. Chen, A. D. Joseph, D. E. Culler, and E. A. Brewer. A composable framework for secure multi-modal access to Internet services from post-PC devices. *Mobile Network Applications*, 7(5):389–406, 2002.

[24] J. Rutkowska. Subverting Vista kernel for fun and profit. Presented at Black Hat USA, 2006.

[25] S. E. Schechter, R. Dhamija, A. Ozment, and I. Fischer. The emperor's new security indicators. In *IEEE Symp. Security and Privacy*, 2007.

[26] R. Sharp, A. Madhavapeddy, R. Want, and T. Pering. Enhancing web browsing security on public terminals using mobile composition. In *MobiSys*, June 2008.

[27] R. Sharp, J. Scott, and A. Beresford. Secure mobile computing via public terminals. In *International Conf. Pervasive Computing*, May 2006.

[28] S. Sheng, B. Magnien, P. Kumaraguru, A. Acquisti, L. F. Cranor, J. I. Hong, and E. Nunge. Anti-phishing phil: the design and evaluation of a game that teaches people not to fall for phish. In *SOUPS*, pages 88–99, 2007.
[29] J. Sobey, R. Biddle, P. C. Van Oorschot, and A. S. Patrick. Exploring user reactions to new browser cues for extended validation certificates. In *ESORICS*, 2008.
[30] M. Stiegler. An introduction to petname systems, Feb. 2005.
[31] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX Security*, Aug. 2009.
[32] Trusted Computing Group. Trusted platform module main specification, Part 1: Design principles, Part 2: TPM structures, Part 3: Commands. Version 1.2, Revision 103, July 2007.
[33] T. Whalen and K. M. Inkpen. Gathering evidence: use of visual security cues in web browsers. In *Graphics Interface*, 2005.
[34] R. Wojtczuk and J. Rutkowska. Xen 0wning trilogy. Invisible Things Lab, 2008.
[35] R. Wojtczuk and J. Rutkowska. Attacking SMM memory via Intel CPU cache poisoning. Invisible Things Lab, 2009.
[36] M. Wu, R. C. Miller, and S. L. Garfinkel. Do security toolbars actually prevent phishing attacks? In *CHI*, 2006.

A Summary of Bumpy Internals

The original Bumpy[6] design [17] is motivated by the prevalence of malware on users' computers, and so the OS (e.g., Windows, Linux) is assumed untrustworthy. Further assumptions inherited from the original Bumpy design include that the remote webserver is uncompromised, and that the SSL certificate provided by the webserver is legitimate and can be extended to integrity-protect the site's favicon. Bumpy enables users to submit sensitive data on web forms without revealing that data to local malware.

§3.1 contains a detailed description of the Bumpy user experience. Here, we summarize the technical underpinnings of the system, specifically the system architecture required to protect user input from a malicious operating system. To do so, Bumpy requires some additional software and (inexpensive and commodity) hardware to enable it to always receive (plaintext) user input before the OS, including: encryption-capable user-input devices (or an encryption-capable interposer, e.g., a small USB device); an isolated and attestable execution environment on the user's computer (e.g., Flicker [16]); a Bumpy software module that executes in this isolated environment, comprising a Pre-Processor and Post-Processor (described below); a trustworthy device with a display to serve as a TM; and Bumpy-aware software on the webserver. These items are in the trusted computing base for Bumpy. Other enhancements that are needed to the software on the client host to work with Bumpy include enhancements to the input-handling logic of the platform OS and an extension to the user's web browser. We stress that these are not in the TCB for Bumpy, however.

We first describe input processing, and then describe how feedback is provided to the user via the TM.

Input Processing. Figure 7 summarizes the flow of user input. Keystrokes are encrypted by an encryption-capable keyboard (or an interposer), and the ciphertext is received by the OS (Steps 1–3). Here, the OS has the opportunity to perform a denial-of-service attack on the inputs, but the OS can always do so by other means (e.g., by powering off or crashing the system, or otherwise preventing meaningful work from being done on the system). A well-behaved OS will invoke a protected execution environment (Step 4), where keystrokes are decrypted and processed by the Bumpy module.

The Bumpy module separates input handling into a Pre-Processor (PreP) and Post-Processor (PoPr). The PreP decrypts incoming keystrokes and tracks whether the current input is sensitive. (Sensitive input begins when the user types the SAS `@@`, and ends when the user provides an input that would cause a *blur* in the web browser GUI; e.g., a tab would be such an event.) Normal input is released to the platform OS decrypted (Step 5), and in this case the user's experience remains unchanged. No further steps are taken for normal input. Sensitive input, however, is queued within the PreP, and decoy input events (e.g., asterisks) are released to the platform OS (also Step 5). When the user finishes providing input to a field that she denotes as sensitive (detected within the PreP by an input event that will cause a *blur* in the web browser GUI), her sensitive input is processed in its entirety. This consists of invoking a (possibly destination-specific) PoPr that will re-encrypt the user's sensitive input for its intended destination (Step 6).

Once the ciphertext containing the user's sensitive input is ready for transmission to its intended destination, a TPM-based attestation [32] is produced. This attestation is used to convince the destination webserver that the user's input was handled with the Bumpy system, the intention being that service providers may be willing to expose additional services to users who are better able to protect their sensitive input. The ciphertext and attestation are handled by an extension to the user's web browser (Step 7) and sent to the remote webserver (Step 8). The webserver will verify the attestation, decrypt the user's input, and process it in accordance with the current application (e.g., a credit card number for an online purchase).

Trusted Monitor. Upon receiving the SAS, the PreP will output an authenticated message for the TM that includes information about the currently active destination website. This information is maintained in the PreP and includes the domain name (Common Name in the site's SSL certifi-

[6] For the rest of this section, Bumpy refers to the original design and not to any of the alternatives proposed in the present paper.

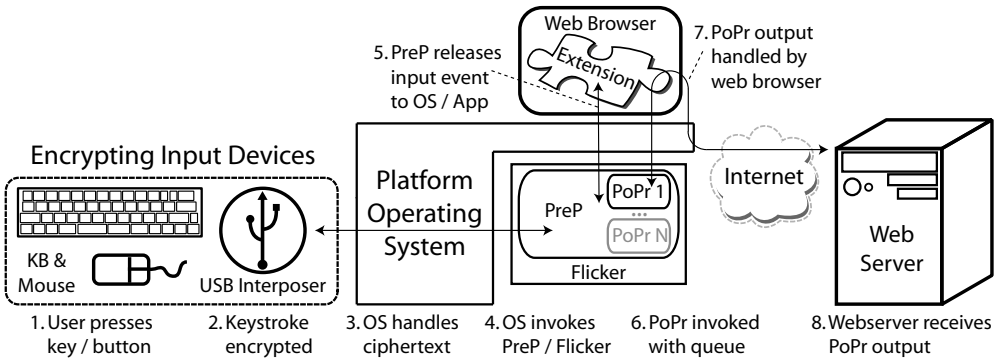


Figure 7. Original Bumpy design for acquiring user input [17]. Steps 1–5 occur for every keystroke or mouse click performed by the user. Steps 6–8 occur only in response to a keystroke or mouse click that the PreP detects will cause a *blur* event in the web browser GUI while the user is entering sensitive data.

cate) and graphic logo (the site's favicon) of the destination webserver. Bumpy reads this information directly from the destination webserver's SSL certificate, and so the domain and graphic logo on the TM are precisely the destination to which sensitive input will be sent. It is the user's responsibility to verify that this is the intended destination.