Operating System Framed in Case of Mistaken Identity

Measuring the success of web-based spoofing attacks on OS password-entry dialogs

Cristian Bravo-Lillo[1]
cbravo@cmu.edu

Lorrie Cranor[1,2]
lorrie@cs.cmu.edu

Julie Downs[3]
downs@cmu.edu

Saranga Komanduri[2]
sarangak@cmu.edu

Stuart Schechter[4]
stus@microsoft.com

Manya Sleeper[2]
msleeper@cmu.edu

Carnegie Mellon University, [1]Engineering and Public Policy, [2]Computer Science, [3]Social and Decision Science
[4]Microsoft Research

Abstract

When asking users to enter credentials, today's desktop operating systems often use windows that provide scant evidence that a trusted path has been established; evidence that would allow a user to know that a request is genuine and that the password will not be read by untrusted principals. We measure the efficacy of web-based attacks that spoof these operating system credential-entry windows to steal users' device-login passwords. We recruited 504 users of Amazon's Mechanical Turk to evaluate a series of games on third-party websites. The third such website indicated that it needed to install software from the publisher that provided the participants' operating system: Microsoft's Silverlight for Windows Vista/7 users and Apple's QuickTime for Mac OS users. The website then displayed a spoofed replica of a window the participant's client operating system would use to request a user's device credentials. In our most effective attacks, over 20% of participants entered passwords that they later admitted were the genuine credentials used to login to their devices. Even among those who declined to enter their credentials, many participants were oblivious to the spoofing attack. Participants were more likely to confirm that they were worried about the consequences of installing software from a legitimate source than to report that they thought the credential-entry window might have appeared as a result of an attempt to steal their password.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Evaluation/methodology, Interaction styles; I.3.6 [**Computing Methodologies**]: Methodologies and Techniques, Interaction Techniques

Keywords

trusted path, user interface, spoofing attack, usable security

CCS'12, October 16–18, 2012, Raleigh, North Carolina, USA.
Copyright 2012 ACM 978-1-4503-1651-4/12/10 ...$10.00.

1. INTRODUCTION

When a user interacts with a computing device, she may actually be communicating with a number of different principals, including the operating system (OS), installed applications, or websites. The security of many user experiences rests on the assumption that there is a *trusted path* from the user to the principal she is communicating with, and that the user can correctly identify (authenticate) this principal. It is assumed that there is a trusted path between the user and the OS for such purposes as authenticating with a shared secret (e.g. a password) and authorizing access to capabilities that may impact the security of the system.

In this paper, we measure the efficacy of web-based attacks that spoof OS windows to trick users of Mac OS X (Mac OS) and Windows Vista/7 (Windows) into entering their device (OS-level) login credentials. The spoofing attacks we tested exploit the fact that these operating systems often request users' device credentials by overlaying windows on top of windows from other principals.

Stealing credentials via spoofing requires two separate forms of social engineering. First, the attacker must give the user *motivation* to enter her credentials. This means creating a task or scenario that the user wants to complete, or believes she needs to complete, and that can only be completed if she provides her credentials. In the extreme, this could be as simple as presenting a window that can be dismissed only by entering credentials, and that the user will want to close. Second, the attacker must spoof the credential-entry interface in place of a genuine interface with sufficient fidelity that the user will *trust* it.

In our attacks, we tried to motivate users to enter credentials by convincing them that they should install software, and that their credentials were required to do so. In many contexts, credentials are indeed required to install software. On Mac OS, installing software requires a privilege elevation, which causes a credential-entry window to appear in the center of the screen. On Windows, User Account Control (UAC [18]) is a protection mechanism that controls the elevation of privilege to allow installations and other sensitive actions. As with Mac OS, it uses a window to verify user intent before elevating. However, unlike Mac OS, its default configuration will only request a username and password if the current user does not have the administrator privilege required for elevation. Furthermore, Windows attempts to differentiate UAC windows from other windows by dimming the contents of the rest of the screen when a UAC window is present.

Windows applications may also request that the OS present the Windows credential-entry experience (CredUI) [14] in situations where the user should re-authenticate. Notably, Microsoft Outlook may request the users' credentials via CredUI, and Microsoft Lync may request credentials through a custom interface, even if the user has already provided these same credentials to login to her PC. CredUI does not dim the portions of the users' screen controlled by other principals.

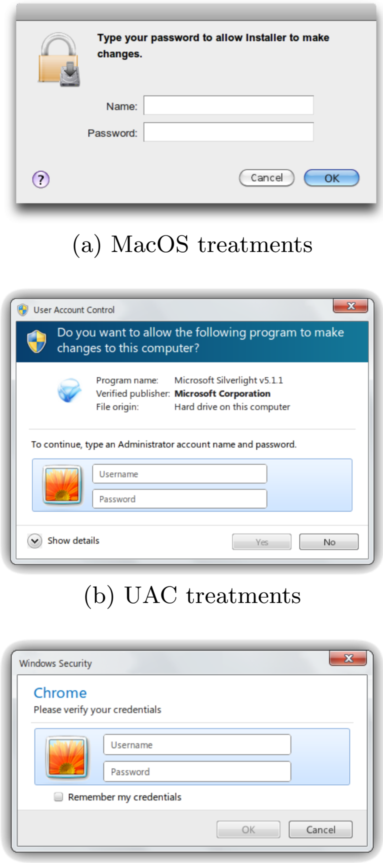
On Mac OS, we spoofed the password-entry window that appears when users need to elevate privileges to authorize software installs (Figure 1a). For Windows users, we tested both a spoofed User Account Control [18] window (Figure 1b) and a Windows CredUI [14] window (Figure 1c).

Studying security behavior is difficult, as individuals' real-world behaviors may differ from how they behave in a laboratory environment. The knowledge that a study pertains to security may cause users to focus on security more than they otherwise would. Thus, asking participants to evaluate warnings directly may lead to answers that are less spontaneous [2]. We used the subterfuge of an Amazon Mechanical Turk task to evaluate third-party online games to create a situation in which real users would encounter a spoofing attack on what appeared to be a third-party site, outside of the control of researchers. While our study design limited our participant pool to users of Mechanical Turk, the methodology increased ecological validity.

We asked 504 workers to evaluate a series of three online games, hosted on third-party websites, for such factors as level of enjoyment and age appropriateness. The third such website, controlled by us, spoofed an OS credential-entry window requiring device-login credentials.

Under our experimental conditions, we found that in over 20% of trials, our most effective attacks yielded passwords that users later admitted that were their genuine device login passwords. Furthermore, when participants who refrained from revealing their credentials were asked why they did so, only 35.3% confirmed that they thought the password-entry interface might be an attempt to steal their passwords (a lack of trust in the interface). Many of the remaining participants may have oblivious to the attack, but simply weren't motivated to enter their credentials to install new software.

The consequences of an attacker obtaining device credentials are different than, and in some cases may be more severe than, an attacker gaining website credentials. They also vary greatly with the security posture of the victim or the victim's organization. If the user's device allows remote access and no firewalls block it, an attacker with the username and password will obtain complete control of the user's device account and will be able to install backdoors and key loggers. If the compromised user has administrative access to the machine, the attacker's keylogger can collect usernames and passwords from other users and the attack can snowball. Enterprise users may be more likely to have remote access restricted by firewalls, but in many cases a Windows domain username and password may be sufficient to allow a new computer to be added to a domain, allowing an attacker to bypass firewalls. Even if compromised credentials cannot be used for remote device access, many enterprises also allow access to web- and smartphone-based email clients using these username and password credentials.


Type your password to allow Installer to make changes.

(a) MacOS treatments

User Account Control

Do you want to allow the following program to make changes to this computer?

Program name: Microsoft Silverlight v5.1.1
Verified publisher: **Microsoft Corporation**
File origin: Hard drive on this computer

To continue, type an Administrator account name and password.

(b) UAC treatments

Windows Security

Chrome
Please verify your credentials

(c) CredUI treatments

Figure 1: Our spoofed credential-entry windows

2. RELATED ATTACKS

The spoofable credential-entry windows we examine in this paper are one instance of a trusted path vulnerability. More familiar examples are phishing of website credentials, in which both emails and websites are spoofed, and scareware, in which attackers spoof infection alerts that appear to come from already-installed trusted software to trick users into installing malware posing as antivirus software.

Felten provides the earliest demonstration of a web spoofing attack, describing it as allowing "an attacker to create a 'shadow copy' of the entire World Wide Web" to observe user behavior and capture user information [9]. The bulk of web spoofing attacks take the form of phishing and its variants, in which attackers spoof email, text messages, voice, and other communications channels to lure victims to spoofed websites. When users login to the spoofed website, their credentials are sent to the attacker. Phishing succeeds because many users cannot, or do not, authenticate websites. Many users instead rely on the content area of the page, assuming that the look and feel of a website are difficult to copy [6]. Such users fail to properly interpret indicators of website authenticity [7]. Increasingly sophisticated spoofing attacks have been implemented, including attacks on more modern browsers [29], attacks that use a graphical element to cover up the SSL lock icon [15], and attacks that spoof the entire browser window, including the certificate functionality [16].

Phishing attacks are similar to the attack in our work in that they use spoofing to steal credentials, motivating a user to login via a URL that leads to the spoofed site and then convincing the user to trust the spoofed site. Scareware attacks are similar to our work in that they often spoof windows that appear to come from trusted client software. Scare tactics motivate users to install fake antivirus software. They create the illusion that the client is already infected with malware. Once installed, the fake antivirus software is used to trick the user into paying to keep the software 'up to date'. Stone-Gross et al. summarize the methods and economics behind such fake antivirus attacks [26].

One of the challenges in understanding the scope of the scareware problem is that fake antivirus software is installed not only through social engineering scare tactics, but also through vulnerabilities, including browser vulnerabilities that allow attackers to perform 'drive-by downloads'. In the one-year period from July 2008 to June 2009, Symantec reports having received 43 million attempts to install rogue security software [27] (Symantec has not released more up-to-date statistics.) Rajab et al. claim that fake antivirus attacks date back as far as 2003, and found that, from January 1 2009 to January 31 2010, such attacks were increasing as a percent of domains containing malware from an incidence rate of 3% to 15% [21]. The fraction of fake antivirus sites that use social engineering as an installation mechanisms also increased to 90% [21].

The best publicly-available statistics on the rate at which users are tricked by scareware come from Cova et al., who discovered servers hosting rogue antivirus campaigns that reported event counts such as the number of users who downloaded the scareware [4]. The researchers discovered that 7.7% of users who received javascript that simulated an antivirus scan initiated a download of the scareware. The actual per-attack success rates were lower as downloads may be aborted before installation commences. Only 5% of machines that presented the fake scan reported back to the attacker's infrastructure that installation was successful (roughly two thirds of the 7.7% that downloaded the scareware.) [3].

3. EXPERIMENTAL DESIGN

We designed our experiment to determine the fraction of users who would enter their passwords into a spoofed OS window, the fraction who would detect that the window was spoofed, and what clues (if any) users looked for to detect if a window was spoofed.

3.1 Experimental procedure

Our experiment mimicked the experience of going to a previously unvisited website and receiving a spoofed installation window designed to steal username and password credentials. The rules of informed consent dictate that we disclose the identity of our research institution before collecting data from users. We thus needed a study design that would allow us to minimize the likelihood that participants' trust in our research institution would cause them to behave less safely than they otherwise would.

We received Institutional Review Board (IRB) approval to create a deception study in which participants were told that their job was to help us evaluate third-party gaming websites. We did so because any trust participants had in our institution should not extend to third-party websites, and so before directing participants to these sites we notified participants that these sites were outside our control. However, the simulated attacks took place on a *confederate* gaming website, which we secretly did control. We use the term *confederate*, because this website was used in much the same way human confederates are employed in other human subjects experiments: presenting the illusion of being a third-party outside the influence of the researchers, creating a situation to which participants could respond (the spoofing attack), and recording participants' behaviors in situations in which participants may have believed themselves to be free from researchers' observation.

3.1.1 Recruitment and screening

Participants volunteered for a Human Intelligence Task (HIT) we had posted on Amazon's Mechanical Turk, as detailed in Appendix A.

Participants had to be at least 18 years old, in the United States while taking the survey, and have an MTurk approval rating of at least 95%. Participants must not have participated in any previous similar study from our lab, including earlier versions (pilots) of this experiment. We used browser-submitted HTTP headers to verify that participants were using either Windows Vista/7 or Mac OS X or higher. We paid $1.00 to each participant who qualified for and completed our survey.

3.1.2 Tasks

Upon accepting the HIT, we redirected participants to a survey site operated in the domain of our research institution. Our survey requested that the participant maximize the browser window so that, when participants reached the third-party website conducting the simulated attack, the appearance of the spoofed window within the browser content area would be less suspicious. We posited that an attacker could use social engineering to convince most users to maximize their browsers.

Our survey presented three game evaluation forms in sequence, each of which asked participants to click on a link opening a third-party online gaming website (a different one for each form), play an online game, and then answer questions about the game. Each form had a link to the game, and participants were instructed to open the link while pressing the ctrl (Windows) or command (Mac OS) key, opening a new browser tab. Participants were asked to play each game for two to three minutes, then close the browser tab and return to the survey form. The evaluation form, given in Appendix B, asked about the game's enjoyability and age-appropriateness, and gave participants the option to report being unable to play a game.

The first two games were real games operated by real gaming websites outside of our control. These two tasks were included to add legitimacy to the subterfuge that we were indeed soliciting evaluations of third-party games. The third and final website, `yourgamefactory.net`, was the confederate website that we secretly operated. It was at this website that participants were told that they needed to install new software to play the game and then saw a spoofed credential-entry window. The exact message of what would be installed and the layout of the spoofed windows was determined by the participant's treatment group.

If participants entered their credentials into the confederate website, they encountered a page that explained that the game had been removed. Regardless, once participants completed evaluating the game, which simply required them to check a box indicating that they couldn't or wouldn't play it, they were presented with our exit survey.

3.1.3 Exit survey

The biggest methodological challenge we faced was to determine if a participant's credentials would have been compromised by our simulated attack. In other words, had they typed their genuine device username and password into the credential-entry window?

Even if it were ethical to send the passwords to our servers without participants' consent, there would be no sure way to tell if the password was legitimate without trying to gain access to the participants' computers—a criminal act. Simply asking participants to tell us if they had entered their genuine credentials seemed likely to result in underreporting. Upon discovering our deception, many would surely be both tempted to deceive us in their response. Furthermore, participants might have legitimately feared that we, as perpetrators of a deception, might misuse their credentials. If they feared their credentials might still be abused, they might justifiably have believed their safest action was to deny that the credentials they revealed were indeed genuine.

While we did not immediately send the contents of the password field to our server, we did send its length l. If the field was non-empty ($l > 0$), we used this value in the exit interview when asking the following 'yes' or 'no' question:

You entered l characters into the password field of the dialog box. This password you entered is stored in your browser but we have not sent your password to our servers. Was the password you entered a real password for an account on your computer?

- ○ **Yes, I did enter a genuine password (we'll immediately delete any records of the password we kept in your browser)**
- ○ **No, I did not enter a genuine password**

If the answer was no, we used a follow-up question to try to identify participants who had entered their genuine passwords but had decided to deny doing so.

Since you did not enter a genuine password into the password field of the dialog box, may we collect the contents of this field for analysis?

- ○ **Yes, since the password I entered is not a genuine password you may send it to your servers.**
- ○ **No, the password I entered was actually a genuine password. Please immediately delete any records of it in my browser and do not send it to your servers for analysis.**
- ○ **No, I have another reason for not wanting the password I entered sent to your servers (please explain)**

We designed this question so that if participants had entered genuine credentials, they would believe the safest option was to admit to doing so. If participants did not enter their genuine credentials, they would only need to enter an arbitrarily short explanation to keep the values they entered private. Participants who admitted that their password was genuine in either of the above two questions would be considered to be *compromised*.

If a participant did not enter a genuine password, we first asked them to use freeform text to "please explain why you didn't enter your password into the password-entry window." We then asked participants to "please indicate which (if any) of the following factors contributed to your decision to not enter your password." Participants could check any number of options, the order of which was randomized for each participant. The options included legitimate concerns, such as not wanting to take the time to install software. The options also included items that should not have been concerns, such as fearing that updated software, published by the company that provided their client OS, would harm their computers. The combination of both types of options was used so that participants would have to think about whether each option really did match their set of concerns. Amongst the list of options was one that stated "I thought that the password-entry window was trying to steal my password." Participants who checked this option are considered *wise* to the spoofing attack. Those who did not are considered oblivious to the spoofing, or *oblivious* for short.

Only after these questions did we disclose the deception and explain the purpose of the study. After doing so, we asked participants such questions as whether they suspected that the window was spoofed and whether they took any actions to test whether the window was authentic. At the end of the survey we asked additional demographic questions.

3.2 Instrumentation

We instrumented the confederate gaming website to record participants' OS type, browser client name and version, screen size, browser viewport size, and the position of the top left corner of browser's viewport relative to the top left corner of the screen. We also recorded participants' mouse movements, clicks within the page content (which included our spoofed window), and number of keystrokes in the username and password field, as well as the timing of each of these events measured in milliseconds.

If the participant tried to submit credentials using the spoofed credential window, we encrypted the contents of the username and password fields using a random one-time-use symmetric encryption key retrieved from, and stored exclusively by, our servers. We stored the ciphertext of the username and password in the client's browser local storage, but discarded the key from the client so that the client could not decrypt the ciphertext. This allowed us to reduce the risk of storing the contents of the username and password fields in the participants's browser, yet gave us the option to transmit the information to the server later if we obtained the participant's consent.

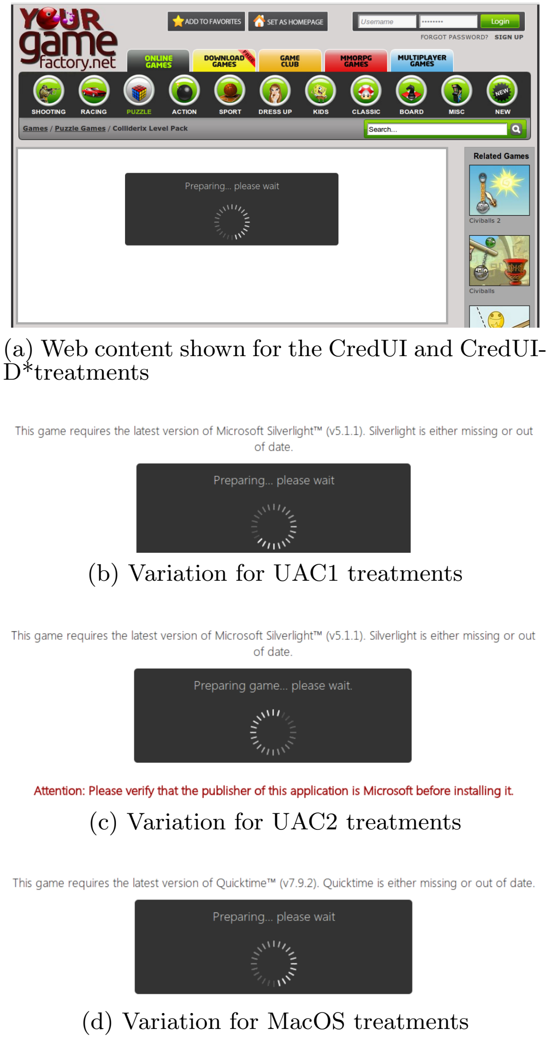
3.3 Implementing spoofed windows

We spoofed the OS window using HTML, CSS, and Javascript. We did not use Flash or other plugins. The window could be moved, though only within the browser content region. For all Windows treatments, we rendered the spoofed window using the default Windows color scheme and implemented a translucent chrome. We also used the same fonts as the genuine Windows dialogs, though because they were rendered on the client, the browser's zoom level could cause them to be rendered the wrong size—a problem that could have been solved by rendering text on the server. Similarly, on Mac OS we tried to match the OS fonts, colors, and other aspects of the genuine window's appearance.

Genuine credential-entry windows often have the username field pre-filled. Since we simulated attacks that did not have access to the username, these fields were left blank in all treatments. We used the generic 'flower' icon to repre-

Treatment	Motivation	Spoofed window
CredUI-D*	Figure 2a	Figure 1c
CredUI		
CredUI-D		
UAC1	Figure 2b	Figure 1b
UAC1-D		
UAC2	Figure 2c	
UAC2-D		
MacOs1	Figure 2d & Figure 3	Figure 1a
MacOs1-D		
MacOs2	Figure 2d	
MacOs2-D		

Table 1: Treatment groups. The credential-entry windows for those treatment groups with a suffix of '-D' had no cancel button or had a disabled cancel button, as well as the window close box disabled.



(a) Web content shown for the CredUI and CredUI-D*treatments

This game requires the latest version of Microsoft Silverlight™ (v5.1.1). Silverlight is either missing or out of date.

Preparing... please wait

(b) Variation for UAC1 treatments

This game requires the latest version of Microsoft Silverlight™ (v5.1.1). Silverlight is either missing or out of date.

Preparing game... please wait.

Attention: Please verify that the publisher of this application is Microsoft before installing it.

(c) Variation for UAC2 treatments

This game requires the latest version of Quicktime™ (v7.9.2). Quicktime is either missing or out of date.

Preparing... please wait

(d) Variation for MacOS treatments

Figure 2: The contents of the confederate website's pre-installation page, over which the credential-entry window appears, were slightly different for each treatment group. Subfigure (a) figure shows the full page, whereas the remaining figures show variations from Subfigure (a).

sent the user account in the UAC dialog, as, if the user had a user-specific icon to represent their account, the attacker would not have access to it.

3.4 Treatment groups

Table 1 shows the 11 treatment groups in our experiment. Participants using a browser on Windows were randomly assigned one of 7 treatments: 3 CredUI treatments and 4 UAC treatments. Participants using a browser on Mac OS were randomly assigned to one of 4 MacOS treatments.

CredUI

In the CredUI treatment, a webpage appeared explaining that the game was being prepared (see Figure 2a) and, after four seconds, the spoofed CredUI window illustrated in Figure 1c was overlaid above it. We placed the name of the browser (e.g. "Internet Explorer" or "Chrome") into the heading of the window to represent the principal requesting the password. Because we did not have a user name in our experiments, the field was left blank and a field label, "Username," was overlaid on top as it would be in the genuine window. (We later discovered that real Windows interfaces actually use the two-word phrase "User name." This small error was common to all Windows treatments.)

UAC

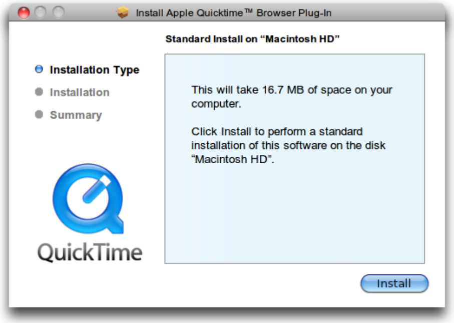
In the UAC1 treatment, the content of the webpage above which the credential-entry window appeared stated that the game required Microsoft Silverlight (Figure 2b). After four seconds a spoofed UAC window appeared asking the participant to consent to the installation of Silverlight and to provide credentials to do so, as illustrated in Figure 1b.

The UAC2 treatment was identical except that the installation window instructed participants to "verify that the publisher of this application is Microsoft before installing it" (see Figure 2c). If users were to check the spoofed window, they would be reassured that Microsoft was indeed listed as the publisher. We hypothesized that users would focus on whether the publisher listed was Microsoft, be reassured that the software was not a threat when they verified that publisher field matched (and was the publisher of their OS), and would thus be less likely to notice that the window in which the publisher field appeared was, in fact, the real threat.

Genuine UAC windows are presented on top of a dimmed desktop and, when credentials are required, the user name is often already filled in. We opted not to dim the region of the screen under our control – the web content region – as we believed that dimming a limited portion of the screen would raise more suspicion than it would dispel.

MacOS

With Mac OS, we simulated the installation of QuickTime as this software is also a common browser add-on and, like Silverlight on Windows, it would be signed by the same company that provided the user's OS. The sequence of events that start an installation on Mac OS begins with a dialog that describes the software to be installed, followed by a request for credentials. Thus, we first showed the webpage that triggered the fake installation (Figure 2d) and, four seconds later, showed a spoofed installation-description dialog (Figure 3). Only after the participant clicked to continue with the installation did we show the spoofed password-entry



Standard Install on "Macintosh HD"

This will take 16.7 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

Figure 3: Spoofed installation-description dialog used exclusively in the MacOS1 treatments

window (Figure 1a). Simulating the installation-description dialog may add realism for Mac OS users who expect to see it before entering their password. However, it also presents the user with an additional opportunity to cancel the installation sequence before the credential-entry window appears. For this reason we omitted several steps that users typically see during the Mac OS installation process, such as selecting the filesystem location in which the application will be stored and consenting to a license agreement.

In the MacOS2 treatment, we did away with the spoofed installation-description window step.

Mac OS puts labels to the left of text fields, rather than overlaying them within, so the username field was completely empty. Mac OS also centers credential-entry windows on the screen—a feature that we did not replicate. We failed to capitalize the 'T' in QuickTime on the "preparing game" webpage, but capitalized it correctly in the spoofed installation-description dialog.

Close-disabled variants

In a pilot of a spoofed UAC window we mistakenly activated the 'Yes' button, which submits the credentials, without waiting for the user to enter characters into the username and password field. Prior research suggests [19] that users will often ignore most of a warning and jump straight to their options. Many participants saw the "Yes" button and pressed it without entering any credentials. From this early mistake, we hypothesized that participants might be more likely to enter their credentials if the option to dismiss the spoofed window was deactivated.

We paired each of the above treatments with a second treatment in which the 'cancel' or 'no' button was removed and the window close box (the 'X' at the top right in Windows) was disabled. These treatments are labeled with the suffix '-D,' and are otherwise identical to their suffixless counterparts.

*CredUI-D**

The final treatment, CredUI-D*, was exactly the same as CredUI-D, except that participants were not asked to maximize their browsers at the start of the study. We included this treatment to determine if it was really necessary for attackers who wanted to spoof an OS window to first trick users into maximizing their browser windows.

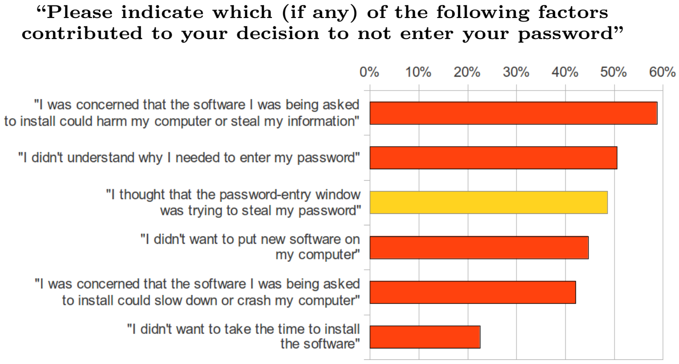


Figure 4: Participants who did not enter any passwords were asked the multiple-answer question above. Participants who picked the third option from top were categorized as 'wise' to the attack; the others were categorized as 'oblivious'.

4. RESULTS

Our results reveal that spoofing attacks are effective, with 18% of participants over all treatment groups providing what they would later admit were valid login credentials for their device. These participants were classified as *compromised*. For reasons discussed in Section 4.2, 6% of participants were *unexposed*, not seeing the spoofed password-entry window.

We asked those participants who did not enter a password to indicate which, if any, factors contributed to their decision not to enter their password. As shown in Figure 4, 48% of those we asked reported that they thought the password-entry window was trying to steal their passwords, and we classified them as *wise* to the attack. 36% did not provide this factor and were classified as *oblivious*.

4.1 Participants

We ran our experiment between January 25 and February 5 2012, and collected results for a total of 504 participants. Not included are an additional 28 participants who were recruited but never visited the confederate gaming website. We also did not include 15 participants who, despite our instructions, responded to the survey from outside the United States.

To identify participants who may not have made conscientious attempts to read and answer questions, we included a multiple-choice question that any participant should have been able to answer correctly: "The power switch on a computer is used to ___?" This approach was inspired by Downs et al. [8]. All but two participants answered this question correctly, suggesting that the majority of participants in our results made a conscientious attempt to read and answer our survey.

Participants were an average of 28 years old (σ =9.6 years), 55% were male, and 78% were caucasian. The top two reported occupations were 'student' (33%) and 'unemployed' (13%). To gauge their level of expertise, we asked five technical questions on topics such as encryption and web security. No participant answered all questions correctly, 11% were able to answer correctly 4 out of 5 questions, and 54% answered one or no questions correctly. 28% reported knowledge of at least one computer programming language. Finally, participants took an average of 17 min 23 secs (σ =18 min 15 secs) to complete the study.

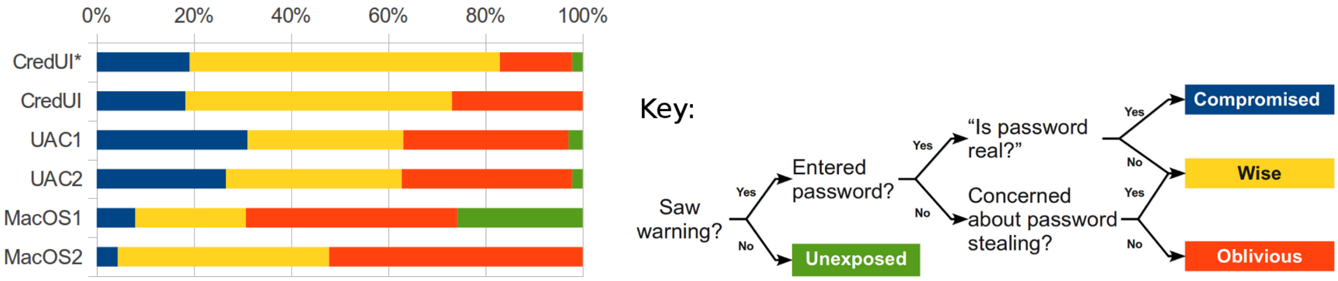


Figure 5: Attack efficacy. *Compromised* participants entered username and password and later admitted that these were valid login credentials for their device. Participants *wise* to the attack didn't admit to entering valid login credentials, and checked the box labeled "I thought that the password-entry window was trying to steal my password." Participants categorized as 'oblivious' were not wise to the attack, but did not fall victim because they had other reasons for not wanting to enter their passwords. *Unexposed* participants may have not seen the credential-entry window (e.g. because they closed the install window presented prior to the credential-entry window in condition MacOS1). With the exception of CredUI-D*, all treatment groups shown above correspond to the merge of the corresponding cancel-enabled and cancel-disabled pair. Disaggregated data can be found in Table 2. The choice of treatment, as shown above, had a statistically significant effect on the outcome.

4.2 Attack efficacy

Of the pool of 504 participants in all treatment groups of our experiment, 142 (29%) entered one or more characters into a spoofed password field. Recall that we used two separate questions to try to compel participants to admit the truth if they had entered their genuine credentials. Of the 142 participants who entered a value into the password field, 38 (27%, or 8% of the total participant pool) denied that the text that they had typed was their genuine device password and, in response to the second question, consented for us to see what they had typed. Another 9 (6% of the participants who typed a password, or 2% of the total participant pool) denied that the password was genuine, but refused to consent for us to see it. The remaining 95 participants (67% of participants who typed a password, or 18% of the total participant pool) admitted to typing their genuine credentials, and all but one did so by answering 'yes' to first of the two questions. (Note, however, that participants may have revised their answer to the first question after seeing the second question.)

The results of each attack are presented in Figure 5, and the disaggregated numbers are presented in Table 2. Our research questions led us to two hypothesis tests, one to determine if close-disabled treatments performed better than their close-enabled peers and one to determine if browser-maximization had an impact on our results. In our analysis we use χ^2 tests to indicate the statistical likelihood that differences between treatment groups were the result of chance. We applied eight tests, and corrected for multiple testing using the Bonferroni method, assuming $\alpha = .00625$ in place of $\alpha = .05$.

In both UAC treatments, the close-disabled treatment pair had a higher compromise rate than a close-enabled treatment pair. In aggregate, the five paired close-disabled treatments had compromise rates (43/215, 20%) slightly higher than their close-enabled peers (43/237, 18%). However, the differences in attack efficacy between conditions with enabled and disabled cancel buttons were not statistically significant ($\chi^2(4) = 1.34$, $p = 0.855$).

To highlight changes between the fundamental designs, we merge close-enabled and close-disabled treatment groups together in Figure 5 (disabling close boxes did not appear to have a significant effect, and was applied with equal frequency to each of the fundamental designs). Over the remaining six treatment groups, we found that the choice of treatment group had a significant effect (after correcting for multiple testing) on both the fraction of participants categorized as *compromised* ($\chi^2(5) = 31.13$, $p < 0.001$) and the fraction categorized as *wise* ($\chi^2(5) = 24.71$, $p < 0.001$).

Some participants were *unexposed* to the spoofed password-entry window because they closed the browser tab before the credential-entry window could appear. Those in the MacOS1 treatments may have aborted installation when the spoofed installation-description dialog was displayed (Figure 3), which came before the credential-entry step.

In every group, at least one participant provided credentials that they later admitted were genuine. For the Windows treatments, the compromise rates in Figure 5 fall between 15% and 35%, with the maximum compromise rate only 2.31× greater than the minimum—a relatively tight range given the sample sizes involved. When combined, participants in all the UAC attack treatment groups had a higher compromise rate (57/192, 29.7%) than those in the two CredUI attack treatments (17/93, 18.3%), though not significant when correcting for multiple testing ($\chi^2(1) = 4.24$, $p = 0.039$).

Many participants who were not compromised may still have been fooled, trusting that the credential-entry window was genuine but failing to enter their credentials due to a lack of motivation. More participants refused to enter their credentials because they were concerned that "the software I was being asked to install could harm my computer or steal my information" (209/356, 58.7%) than because they were afraid to enter their credentials (173/356, 48.6%). Similarly, more participants withheld their credentials not because they detected the spoofing attack, but because they were not motivated to install new software (80/356, 22.5%). Furthermore, a relatively high proportion of participants across all treatments (180/356, 50.6%) volunteered that they did not enter their credentials because they did not know what credentials to enter.

A greater proportion of participants in the two CredUI treatments were wise to the attack (51/93, 54.8%) than those in the four UAC treatments (67/192, 34.8%, $\chi^2(1) =$

10.27, $p = 0.001$, significant when corrected for multiple testing.)

A disproportionately small fraction of participants were compromised by the MacOS2 treatments (4/92, 4.4%), which did not spoof an installation-description window before the credential-entry window, and a disproportionately large fraction were wise to it (40/92, 43.5%). In contrast, the MacOS1 treatments did present an installation-description window. This presented participants who did not want to install QuickTime an opportunity to abandon the process before seeing the credential-entry window, so 26% of participants were not exposed to it (hence *unexposed*). Even accounting for those unexposed, more participants were compromised by the treatments that included the spoofed installation-description window (8/101, 7.9%) than without it (4/92, 4.4%).

While the aggregate compromise rates for participants in the Mac OS treatments were lower than for the Windows treatments, it would be premature to conclude that Mac OS users are less vulnerable to spoofing; we put more effort into tuning our Windows attacks and the treatments targeted different software to install (Silverlight vs. QuickTime). Even when we did present an installation-description dialog, we skipped a number of steps in the installation ritual.

The difference between the rates at which participants in the Mac OS treatments were wise to spoofing (63/167, 37.7%) and those in the Windows UAC treatment groups (67/192, 34.9%) were well within the margin of error ($\chi^2(1) = 0.31$, $p = 0.578$).

The CredUI-D* treatment, in which participants were not asked to maximize the browser window at the start of the survey, did no worse than the identical treatment in which participants were asked to do so (CredUI-D). Of those in the CredUI-D* treatment, our instrumentation indicates that 78% already had their browser sized to consume at least 80% of the screen's area. It's possible that convincing users to maximize browser windows may raise more suspicions than it dispels. In future experiments we may consider removing the window-maximization step, though we cannot say with confidence that this will increase attack efficacy. UAC windows are bigger than CredUI windows, and, therefore, failing to maximize a small browser window might be more likely to cause a user to become wise to a UAC-based attack than a CredUI-based attack. While we believe that convincing users to maximize their browser windows in advance of an attack poses little challenge to social engineers, our data suggests that doing so may be unnecessary.

Finally, after we disclosed the deception to our participants, we asked them whether they "know that the password-entry window was actually mimicked by the website, and not a real password request from [their] operating system." The answers to this question ranged from 'I was completely sure that the password entry wasn't real' to 'I never suspected'. While no 'compromised' participants reported to be completely sure about the deception, between 16% and 25% of participants who were not 'compromised' reported that they were sure of the deception.

4.3 Drop-out rates

During the study, 136 participants began answering the survey but dropped out before finishing. We have only partial information about who these participants were and what they did, but our instrumentation allowed us to know at what point they dropped out of the study. Out of the 136 participants who did not complete the study, 6 were from outside the United States and would not have been included if they finished. 18 participants dropped during the consent form (that is, they did not reach the first game), 56 dropped during the first game (they did not see the second game), 23 dropped during the second game, and 33 dropped during the third game. Out of these 33, two participants skipped all the games (they checked the "I could not play this game" checkboxes), 25 evaluated two games and dropped during the third game (we don't have evidence that these participants visited the gaming website or saw our spoofed dialog), and finally 6 participants reached the third game, saw our simulated dialog and interacted with it. Two of them did not enter a password, returned to our survey, answered a few more questions, and then dropped out; three of them entered a password and never returned to the survey; and finally one last participant entered a password, returned to the survey, said that his or her password was real, and then dropped out. We included these last 6 participants in our analyses reported above.

4.4 Reasons for suspecting spoofing

We asked participants with suspicions to explain them to us. For many participants, especially Mac OS users, the empty username field was a source of suspicion. If an attacker can obtain the user's username, the credibility of that spoofed window, and the compromise rate, would likely rise.

Some participants found ways to test whether the spoofed credential-entry window was real, sometimes using techniques we had not anticipated. One Mac OS participant used Exposé, which reveals the set of OS-level windows, to see that the credential-entry window was only present within the browser. Other Mac OS participants noticed that we had failed to center the spoofed dialog on their screens and knew that the genuine dialog is centered. Some participants in the Windows UAC treatment group were familiar with UAC and knew their computers were configured such that they would not request a password to complete an elevation. In the words of a participant, *"when i installed silverlight before it didn't ask for a password."* Similarly, other participants were aware that their account did not have administrator rights. In the words of another participant, *"Microsoft Silverlight is not installed on my computer, and I am not the 'Administrator' of this computer so I did not know the password. If I had known the password, I probably would have entered it and not thought anything about it."*

Removing paths through which users can close a credential-entry window also raised suspicions among some participants on both Mac OS and Windows. For example, a Mac OS participant reported that the spoofed window *"didn't have the clean Apple 'look' [and] didn't have an available cancel button."* Changing the appearance of the window close box and removing the cancel button may have provided a visual clue that the window was fake. Disabling such functionality more subtly (e.g. by making active-looking buttons non-functional) might increase compromise rates, as the only users who would be alerted to the difference would be those who had already decided to close the window (and thus would not be entering their credentials). Some fraction of those users might then decide to enter their credentials if they believed it was the only way to dismiss the window.

Finally, a handful of participants, familiar with research

	CredUI-D*		CredUI		CredUI-D		UAC1		UAC1-D		UAC2		UAC2-D		MacOS1		MacOS1-D		MacOS2		MacOS2-D	
compromised	9	19%	11	21%	6	15%	15	27%	17	35%	10	21%	15	32%	4	8%	4	8%	3	6%	2	4%
oblivious	19	40%	24	45%	18	45%	22	40%	18	37%	18	38%	18	38%	22	43%	22	42%	31	65%	18	39%
wise	18	38%	18	34%	16	40%	15	27%	14	29%	18	38%	14	30%	15	29%	10	19%	14	29%	26	57%
unexposed	1	2%	0	0%	0	0%	3	5%	0	0%	2	4%	0	0%	10	20%	16	31%	0	0%	0	0%
Total	47	100%	53	100%	40	100%	55	100%	49	100%	48	100%	47	100%	51	100%	52	100%	48	100%	46	100%

Table 2: Disaggregated data for the attack rates, per condition.

studies, were not fooled by the study scenario and the subterfuge of the fake site. One wrote: *"I'm taking a psychology study. I just figured it was part of the study."*

4.5 Follow-up experiment

We performed a second experiment to more tightly bound our estimate of the efficacy of one of our most effective attacks: UAC1. We collected data for 199 participants during two solicitation periods on July 20 and 25 of 2012—a fourfold increase in the number of participants per treatment. In both sessions our participant quotas were met within a matter of a few hours, in contrast to our earlier study which was offered over a greater diversity of times-of-day. Participants' demographic data were virtually identical in both experiments. In the second experiment participants were an average of 29 years old with a $\sigma = 9.7$ years (vs. 28 years old, $\sigma = 9.6$), 53% were male (55% in the original), 77% were caucasian (78% in the original), and the top two reported occupations were again 'Student' (28% vs. 33% in the original) and 'Currently Unemployed' (16% vs. 13% in the original). Participants took in average 19 min 57 sec to complete the study with a σ =8 min 26 sec, vs. 17 min 23 sec with a σ =18 min 15 sec.

Out of 199 participants, 52 entered at least one character, and 41 of them (21% of the total) later admitted it was a real password in either the first or the second question described earlier. Of those who did not proceed to enter characters into the password field and were asked why, 63 (32% of the total) indicated password-theft as a concern, causing us to categorize them as *wise*. The remaining 95 (47% of the toal) were deemed *oblivious* to the attack. The most frequently invoked reason for not entering a password was 'concern that the software could damage their computers', checked by 104 participants (52% of the total), and 'not wanting to install new software', checked by 86 participants (43% of the total).

In this follow-up experiment, 24 participants did not complete the survey, 6 of them being from outside the US. Of the 18 remaining, 14 dropped before getting to the third game. Only one of the four remaining participants returned to the survey after having seen the spoofed dialog; we don't have evidence that the other three actually saw the dialog.

The compromise rate in the follow-up experiment (21%) was lower than for the same treatment in the original experiment (27%), although the difference is not significant ($\chi^2(1) = 0.429$, $p = 0.5125$). Figure 6 displays the 95% confidence intervals for the compromise rates observed in this experiment.

5. LIMITATIONS

As with any experiment, our study has limitations that may cause our results to differ from the results of a real attack, including the 5% attack efficacy results for scareware campaigns previously reported by Cova et al. [3, 4].

Our participants were drawn from the population of users of Mechanical Turk who accepted our HIT. This population may differ in important ways from the populations targeted in certain attacks. For example, an attacker targeting a software security company might compromise a smaller proportion of users than were compromised in our experiments, as such individuals may be more likely to detect spoofed windows. Mechanical Turk users may be more or less likely to be using personal (as opposed to a work) accounts and thus the vigilance with which they protect their credentials may differ from the populations targeted in real attacks.

Some factors may have made our simulated attacks more likely to result in a compromise than a real spoofing attack. For example, participants may have recognized the name of our institution in the initial consent disclosure and assumed that researchers would not direct them to an unsafe third-party site. Participants may also have mistyped their credentials but reported that they had entered their valid credentials. Additionally, convincing users to maximize their browser, or doing it for them, may be essential to achieving the compromise rates we saw. However, we did not see evidence of this when we compared the CredUI-D* and CredUI conditions. Attackers may be unable to convince users to maximize windows without causing suspicion.

It is also possible that attackers could achieve compromise rates much higher than those we saw in our experiments. An attacker who could provide a more compelling scenario for entering credentials might be able to compromise many of those users who would not be compromised by our treatments. A real attacker need not repeat mistakes we made when learning to spoof these interfaces, such as using the word 'Username' instead of 'User name' in Windows and failing to center the Mac OS credential-entry dialog. A real attacker spoofing an installation of Silverlight might put a Silverlight object on the page to detect whether Silverlight was already installed and to identify the current version.

In considering the results of our study, one must also con-

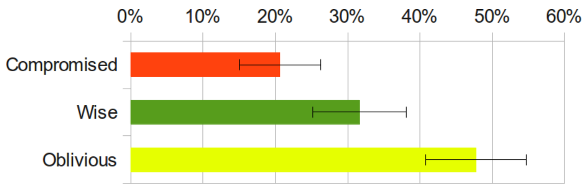


Figure 6: Attack efficacy for second experiment, along with 95% confidence intervals. 20.6±5.6% of participants were *compromised*, 31.7±6.5% were *wise* to the attack, and 47.7±6.9% were *oblivious* to the attack.

sider that the consequences of a compromise vary widely based on what they are used for. If the user's device blocks all forms of remote access, the compromise of device credentials may be of no consequence. If the credentials are for the user's account on an enterprise network, and that enterprise offers remote access to the network, computing, and services (e.g. email, payroll, etc.), the consequences could be significant. If the user employs the same credentials for other accounts, the consequences may extend even further.

6. RELATED DEFENSES

Operating system designers have been aware of the need to defend against trusted-path vulnerabilities since at least as far back as the early 1970's when Saltzer and Schroeder presented the need for a 'secure' path in the context of a scenario in which a user grants permissions (capabilities): "one thing is crucial–that there be a secure path from Doe, who is authorizing the passing of the capability, to the program, which is carrying it out." More recently, Ka-Ping Yee described trusted path as requiring "an unspoofable and faithful communication channel between the user and any entity trusted to manipulate authorities on the user's behalf." Yee highlighted the secure attention sequence in Windows (ctrl-alt-delete) as an example solution to the trusted path problem for credential entry [31].

Many of the defenses that protect users from spoofing attacks today rely on detecting bogus emails and blacklisting software and websites; they do not address the underlying trusted path problem. Such defenses are necessary because preventing spoofing not only requires a technology to support a trusted path, but a change in user behavior to avoid untrusted paths. Users must unlearn the habit of providing credentials into windows they they cannot authenticate. This will require time and a clear set of rules that users can apply to reliably differentiate the OS from other principals.

There are three major categories of solutions to establish trusted paths: dedicated IO, visualizations of shared secrets, and secure attention sequences.

6.1 Dedicated IO

One way to establish a trusted channel between the user and an OS is to dedicate specific hardware, or portions of hardware, to be used exclusively for that channel. For example, a device could dedicate a screen and separate keypad for use in authentication, as is sometimes done in payment systems. Others employ a separate input and output device that the user may already have. For example, Parno et al.'s Phoolproof Phishing scheme employs the user's mobile phone to externally confirm websites when entering a password [20] and IBM's Zone Trusted Information Channel provides an external trusted path for banking operations [24].

Enabling users to communicate securely with a single principal need not necessarily require both a dedicated input and output device. A dedicated output, such as an LED or dedicated screen region, may be sufficient to indicate when a trusted path is present. A dedicated input device may be sufficient to force a trusted path to be established, or may itself be used for the sole purpose of entering credentials.

Many systems attempt to establish trusted paths by dedicating pixels within a window to host trust indicators that indicate the presence of a trusted path. For example, the "chrome" region in browsers is the portion of the browser window that is not controlled by the website being rendered, and has been used to host indicators that activate when a connection is secure or that display the domain name of a website. However, users can still be confused about whether a window is real or fake. For example, Jackson et al. demonstrated that users will trust spoofed chrome elements that appear in a browser window that is itself spoofed, rendered within the content region of a genuine browser window [13]. This is known as a picture-in-picture attack.

Operating systems sometimes use visual cues to differentiate active windows (those 'in focus') from inactive ones, in part to defend against picture-in-picture attacks. For example, some systems render the frames of foreground windows to appear darker than background windows. An astute user might notice that our window in a picture-in-picture attack remain active. Secure windows management systems EROS [25] and Nitpicker [10] dim all windows except the application currently in use and clearly label windows to help prevent users from accidentally entering information into the incorrect application. The results of our experiment raise doubts as to whether dimming the screen is an effective way to establish a trusted path, and if an entire window can be spoofed, the labels inside can be as well.

6.2 Visualizations of shared secrets

While operating systems allow other principals to use the screen, they can usually ensure that they themselves can render data to the screen without it being intercepted by other applications. Thus, if the operating system and user share a secret, the OS can display this secret with reasonable confidence that other principals will not learn it. Shared secret schemes work much in the same way a dedicated output device does, but instead of lighting up a dedicated set of pixels to signal a trusted path, the OS renders a representation of the shared secret.

Tygar and Whitten propose "requir[ing] the consumer to personalize the appearance of the software at the time the trust relationship is formed" for this purpose [28]. Similarly, Adelsbach et al. suggest personalizing security indicators in the browser interface [1]. Dhamija and Tygar's Dynamic Security Skins tool displays a user-selected photograph in windows requesting or providing security information, allowing users to verify that the window was produced by the web browser and not a website [5]. In Herzberg and Jbara's Trustbar, users assign names or logos for each website, and these are later shown to confirm that the users are again at the same website [11].

Other solutions use secrets that are not directly controlled by the user. Ye et al. present a colored border for windows to indicate when these windows are controlled by the browser. The border format dynamically changes to match a browser-controlled metadata window [30].

The security of shared secret schemes rests on the assumptions that users will be able to recognize the shared secret, notice when the shared secret is absent, and realize there is no trusted path when the shared secret is absent. Shared secret schemes may be attacked by convincing users to disregard an invalid or missing secret. For example, Schechter et al. demonstrated an attack against the Passmark shared-secret scheme used for online banking in which users were told that their shared secret was temporarily unavailable due to system maintenance [23].

6.3 Secure attention sequences

Just as shared secrets leverage the operating systems' ultimate control over output devices, secure attention sequences leverage their ability to capture and prioritize input events. For example, on Windows the key combination of ctrl-alt-delete is captured by the operating system, and triggers the establishment of a trusted path to the OS, regardless of what applications are running. Since Windows NT, the OS has required that users unlock their computer with this sequence of keys, a *secure attention sequence*, before logging into the device. This sequence stops the execution of other processes, ensuring the existence of a trusted path for the authentication process [12]. Alas, Windows does not explicitly tell users not to enter their passwords without typing the secure attention sequence, and legitimate applications often ask users to do so.

A number of phishing prevention mechanisms have been used as secure attention sequences, including a 2005 proposal by Ross et al. [22]. Libonati et al. performed a field study to measure the efficacy of secure attention sequences in protecting web logins. No mechanism came close to being foolproof, even though participants in the study knew that they were being tested on their abilities to protect themselves from attacks, and given incentives to protect their passwords [17].

In summary, solving the trusted path problem is daunting. Providing dedicated input or output devices for authentication is impractical: It is costly, consumes device space, and would require a redesign of myriad devices. Trusted chrome has proven too easily spoofable. Establishing a trusted path via users' existing mobile devices for authentication simply passes the buck onto another general computing platform, which may also be vulnerable to spoofing attacks. Users forget to enter secure attention sequences when they don't appear to be necessary.

7. A PATH FORWARD?

The challenge in developing strategies to address the trusted path problem is that one cannot use lab studies, or even short-term field studies, to prove these strategies are resilient to attack. Users are habituated to security rituals over time through repeated conditioning. To study how participants respond to attacks, researchers must study participants who are already conditioned. In other words, proving that a trusted path ritual will resist real-world attacks requires deploying the ritual into the hands of users who will be relying on it to do so.

Given the extreme costs of establishing the security of a trusted path ritual, we think it's essential to learn as much as possible from what isn't working today. Relying on subtle cues to establish that a Window belongs to the OS does not seem to work.

The failures of individual trusted path mechanisms suggest that the problem is unlikely to be addressed adequately by any single mechanism. Rather, future work may focus on rituals that combine mechanisms. For example, consider rituals in which users must first enter a shared attention sequence and then expect to see a visual shared secret. These two mechanisms may complement each other: the expectation of a visual shared secret may make it harder for an attacker to trick the user into entering a password (or performing a security-sensitive action) without first providing the secure attention sequence to make the visual shared secret appear. The secure attention sequence may make it harder to trick the user into believing the visual shared secret is unnecessary—the user need only enter the secure attention sequence to check if it can be made to appear.

Finally, even if a trusted path ritual can be created, the existence of the ritual alone will be insufficient to protect users. So long as users are regularly asked to provide credentials or perform security-critical actions via other paths, they will be habituated to do so when the next attack comes.

8. CONCLUSION

Only a minority of participants in our study recognized that a spoofed operating system credential-entry window was an attempt to steal their passwords. In our most effective attacks, more than 20% of Windows users entered usernames and passwords into a spoofed UAC window, later admitting that these were genuine device-login credentials. Providing a trusted path through which users can enter credentials securely is not in itself sufficient to prevent such attacks. Rather, operating systems must also forbid the collection of device-login credentials through less secure paths (e.g. spoofable windows), lest users become habituated to entering credentials when straying off the trusted path.

Acknowledgements

The authors want to thank John Douceur (Microsoft Research), Serge Egelman (UC Berkeley), David Molnar (Microsoft Research), Rob Reeder (Microsoft), Adam Shostack (Microsoft) and the anonymous reviewers for their helpful reviews and suggestions on early versions of the paper. This research was funded in part by NSF grants CNS0831428, CNS1116934, and DGE090365.

9. REFERENCES

[1] Adelsbach, A., Gajek, S., and Schwenk, J. Visual spoofing of SSL protected web sites and effective countermeasures. *Information Security Practice and Experience* (2005), 204–216.

[2] Bravo-Lillo, C., Cranor, L. F., Downs, J., and Komanduri, S. Bridging the gap in computer security warnings: A mental model approach. *IEEE Security & Privacy Magazine 9*, 2 (Mar. 2011), 18–26.

[3] Cova, M. Personal corresponence, May 5, 2012.

[4] Cova, M., Leita, C., Thonnard, O., Keromytis, A. D., and Dacier, M. An analysis of rogue AV campaigns. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection (RAID 2010)* (Sept. 2010), pp. 442–463.

[5] Dhamija, R., and Tygar, J. D. The battle against phishing: Dynamic security skins. In *Proceedings of the 2005 Symposium on Usable Privacy and Security* (New York, NY, USA, 2005), SOUPS '05, ACM, pp. 77–88.

[6] Dhamija, R., Tygar, J. D., and Hearst, M. Why phishing works. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2006), CHI '06, ACM, pp. 581–590.

[7] Downs, J. S., Holbrook, M. B., and Cranor, L. F. Decision strategies and susceptibility to phishing. In *Proceedings of the Second Symposium on Usable Privacy and Security* (New York, NY, USA, 2006), SOUPS '06, ACM, pp. 79–90.

[8] Downs, J. S., Holbrook, M. B., Sheng, S., and Cranor, L. F. Are your participants gaming the system?: Screening mechanical turk workers. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems* (New York, NY, USA, 2010), CHI '10, ACM, pp. 2399–2402.

[9] Felten, E. W., Balfanz, D., Dean, D., and Wallach, D. S. Web spoofing: An Internet con game. In *20th National Information Systems Security Conference* (Oct. 1996).

[10] Feske, N., and Helmuth, C. A nitpicker's guide to a minimal-complexity secure GUI. In *Proceedings of the 21st Annual Computer Security Applications Conference* (Washington, DC, USA, 2005), IEEE Computer Society, pp. 85–94.

[11] Herzberg, A., and Gbara, A. Security and identification indicators for browsers against spoofing and phishing attacks. Cryptology ePrint Archive, Report 2004/155, 2004. `http://eprint.iacr.org/`.

[12] Initializing Winlogin, 2012. `http://msdn.microsoft.com/en-us/library/windows/desktop/aa375994(v=vs.85).aspx`.

[13] Jackson, C., Simon, D. R., Tan, D. S., and Barth, A. An evaluation of extended validation and picture-in-picture phishing attacks. In *Proceedings of the 11th International Conference on Financial Cryptography and 1st International Conference on Usable Security* (Berlin, Heidelberg, 2007), FC'07/USEC'07, Springer-Verlag, pp. 281–293.

[14] Kerr, K. Defend your apps and critical user info with defensive coding techniques. *MSDN Magazine* (Nov. 2004). `http://msdn.microsoft.com/en-us/magazine/cc163883.aspx`.

[15] Lefranc, S., and Naccache, D. Cut-&-paste attacks with java. In *Proceedings of the 5th International Conference on Information Security and Cryptology* (Berlin, Heidelberg, 2003), ICISC'02, Springer-Verlag, pp. 1–15.

[16] Li, T.-Y., and Wu, Y. Trust on web browser: Attack vs. defense. In *Applied Cryptography and Network Security*, J. Zhou, M. Yung, and Y. Han, Eds., vol. 2846 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2003, pp. 241–253. 10.1007/978-3-540-45203-4_19.

[17] Libonati, A., McCune, J. M., and Reiter, M. K. Usability testing a malware-resistant input mechanism. In *Proceedings of the 18th Annual Network & Distributed System Security Symposium (NDSS11)* (Feb. 2011).

[18] Microsoft Corporation. What is user account control? `http://windows.microsoft.com/en-US/windows-vista/What-is-User-Account-Control`.

[19] Nodder, C. Users and trust: A microsoft case study. In *Security and Usability: Designing Secure Systems That People Can Use*, L. F. Cranor and S. L. Garfinkel, Eds., first ed., Theory in practice. O'Reilly Media, Inc., Sebastopol, CA, USA, 2005, ch. 29, pp. 589–606.

[20] Parno, B., Kuo, C., and Perrig, A. Phoolproof phishing prevention. In *Proceedings of the Financial Cryptography and Data Security 10th International Conference* (2006), FC'06.

[21] Rajab, M. A., Ballard, L., Mavrommatis, P., Provos, N., and Zhao, X. The nocebo* effect on the web: An analysis of fake anti-virus distribution. In *Proceedings of the 3rd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More* (Berkeley, CA, USA, 2010), LEET'10, USENIX Association, pp. 3–3.

[22] Ross, B., Jackson, C., Miyake, N., Boneh, D., and Mitchell, J. C. Stronger password authentication using browser extensions. In *Proceedings of the Proceedings of the 14th Usenix Security Symposium* (Aug. 2005).

[23] Schechter, S. E., Dhamija, R., Ozment, A., and Fischer, I. The emperor's new security indicators. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 51–65.

[24] "Security-on-a-Stick" to protect consumers and banks from the most sophisticated hacker attacks, October 2008. http://www.zurich.ibm.com/news/08/ztic.html.

[25] Shapiro, J. S., Vanderburgh, J., Northup, E., and Chizmadia, D. Design of the EROS trusted window system. In *Proceedings of the 13th Conference on USENIX Security Symposium* (Berkeley, CA, USA, 2004), SSYM'04, USENIX Association, pp. 12–12.

[26] Stone-Gross, B., Abman, R., Kemmerer, R. A., Kruegel, C., Steigerwald, D. G., and Vigna, G. The underground economy of fake antivirus software. In *Workshop on Economics of Information Security (WEIS)* (June 2011).

[27] Symantec Corporation. Symantec report on rogue security software, Oct. 2009.

[28] Tygar, J. D., and Whitten, A. WWW electronic commerce and Java trojan horses. In *Proceedings of the Second USENIX Workshop on Electronic Commerce* (Berkeley, CA, USA, 1996), vol. 2, USENIX Association, pp. 15–15.

[29] Ye, E., Yuan, Y., and Smith, S. Web spoofing revisited: SSL and beyond. Tech. Rep. TR2002-417, Dartmouth College, 2002.

[30] Ye, Z. E., Smith, S., and Anthony, D. Trusted paths for browsers. In *Proceedings of the 11th USENIX Security Symposium* (2002), pp. 263–279.

[31] Yee, K.-P. User interaction design for secure systems. In *Proceedings of the 4th International Conference on Information and Communications Security* (London, UK, 2002), ICICS '02, Springer-Verlag, pp. 278–290.

APPENDIX

A. PARTICIPANT SOLICITATION

Researchers at Carnegie Mellon University are conducting a set of brief surveys about online games. You will have to play three online games, and then answer a short survey giving us your opinion about each game. The whole survey should take you about 20 minutes. We will pay you $1.00 for your participation.
Requisites to participate:

1. You must be 18 years old or older.
2. You must be in the United States while you take the survey.

3.w [*shown only to users Windows clients*]
You must use Microsoft Windows Vista or Windows 7. We will not pay you if you use another operating system, or an older version of Microsoft Windows (like Windows XP). You don’t have to use MS Internet Explorer, but if you do you must use Internet Explorer 8 or higher.

3.m [*shown only to users of MacOS clients*]
You must use Apple MacOS to participate. We will not pay you if you use another operating system.

3. You cannot take this survey twice. Please click here to check if you have taken this survey before.

To be paid, follow these steps:

1. Go to: `http://saucers.cups.cs.cmu.edu/yacot/mnt/wtk/survey.php?i=workerID`
2. After completing the survey you will receive a confirmation code in the last page. Enter the code in the box below and we will approve your payment. Please do not enter the code more than once. If you are not sure about having entered the code correctly, please send us a message and we will solve the problem as soon as possible.

B. EXAMPLE GAME EVALUATION FORM

Instructions to evaluate the game:

1. While pressing CTRL/Command on your keyboard click on the link below to open the game in a new tab of your browser.
2. Click on the button ”Click to play online” on the left of your screen. Wait for the game to load.
3. When the game has loaded completely, play the game ”Mars Buggy Online” for about 2 to 3 minutes.
4. Return to this survey to answer the questions below.

Assigned game N: Mars Buggy Online
http://www.gametop.com/online-free-games/mars-buggy-online/
(Press CTRL/Command while clicking this link)
Attention: The website whose URL appears above is external to this study. Our researchers **do not** control its content.

If you are not able to download or install the game above, please check the box below and then click ’Next’ on the bottom of the page. You will be assigned a new game to evaluate.
□ I was not able to download or install the game, please assign me another game to evaluate.
Please tell us briefly why you were not able to play the game: (required open text)

Please enter here a one-sentence description of the game you played (between 10 and 50 words): (optional open text)

Please answer the following questions about the game you played:

Have you ever played this game before?
○ Yes
○ No

Do you think this game is appropriate for children between 4 and 8 years old?
○ Yes
○ No

Do you think this game is appropriate for pre-teenagers between 9 and 12 years old?
○ Yes
○ No

Do you think this game is appropriate for teenagers between 13 and 17 years old?
○ Yes
○ No

Do you think this game is fun?
○ Yes
○ No